Universitetet i Oslo
Institutt for Informatikk

Hovedfagsoppgave
Studieretning Kommunikasjonssystemer
Våren 2003

Peer-to-Peer fildeling for mobiltelefoner

Forfatter:	**Henrik Heiestad**
E-post:	henrik@heiestad.no
Ekstern veileder:	**Sigrid Steinholt Bygdås**, forsker Telenor FoU
Intern veileder:	**Kjell Åge Bringsrud**, Førsteamanuensis UiO
Levert:	1.mai 2003

Forord

Denne hovedfagsoppgaven er skrevet i samarbeid med forskningsprogrammet Anarkistiske Nettsamfunn ved Telenor A/S Forskning og Utvikling. Oppgaven omhandler temaet "Peer-to-Peer (P2P) fildeling for mobiltelefoner" og er en del av forskningsgruppens arbeid med P2P-plattformen JXTA. Denne oppgaven har vært med på å belyse temaet P2P-fildeling i en mobil kontekst. Dette har blitt gjort gjennom en prototyp som realiserer P2P-fildeling på mobiltelefoner.

Motivasjonen til oppgaven fikk jeg da min studiekamerat Tommy Gagnes tipset meg om JXTA og JXTA for Java 2 Micro Edition (JXME) en gang i slutten 2001. Etterhvert kom jeg i kontakt med Andre Mlonyeni og Sigrid Steinholt Bygdås ved Telenor FoU. Disse tilbød meg en oppgave som passet meg veldig godt.

Det har vært givende å få jobbe innen et så spennende felt som P2P for mobiltelefoner. Ingen har tidligere laget en P2P fildelingsapplikasjon for mobiltelefoner. Til tross for mye frustrasjon underveis har det lyktes å komme i mål til planlagt tid, mye takket være veileder Sigrid Steinholt Bygdås. Resultatet er en fullt brukbar prototyp, mange nye erfaringer og ny kunnskap innen området P2P fildeling for mobiltelefoner.

Oppgaven er skrevet på norsk. Det har derfor vært et mål å bruke mest mulig norske ord og uttrykk. Enkelte ord og uttrykk lar seg dessverre ikke oversette på en tilfredstillende måte. I disse tilfellene bruker jeg de engelske ordene, og forklarer heller hva som ligger i de.

Takk til
Jeg vil benytte anledningen til å takke en del personer som har hjulpet meg gjennom prosjektet:
Takk til sivilingeniør Tommy Gagnes som tipset meg om JXTA, som jeg har hatt mange faglige diskusjoner med og som har hjulpet til med gjennomlesning av oppgaven.
Takk til veileder Sigrid Steinholt Bygdås som gjennom hele prosjektet har hjulpet meg med praktiske og faglige utfordringer. Takk for hjelp til skriving av oppgaven.
Takk til Andre Mlonyeni, leder for forskningsprogrammet Anarkistiske Nettsamfunn, for at jeg fikk lov til å skrive oppgave i samarbeid med forskningsprogrammet.
Takk til intern veileder Kjell Åge Bringsrud ved Institutt for Informatikk.
Takk til min kone Nina Elisabeth Heiestad som har vært en støtte gjennom hele skolegangen.

Innholdsfortegnelse

Abstract

Keywords: *Peer-to-Peer, filesharing, cellular phones, JXTA, J2ME*

This thesis shows that it is possible to realize a Peer-to-Peer filesharing application on limited devices like cellular phones. The Peer-to-Peer model is especially appropriate to the characteristics of wireless devices. We believe that there is a great potential in P2P filesharing applications for cellular phones. The application, *MOBster*, is built on top of Sun's Peer-to-Peer platform called JXTA. JXTA for Java 2 Micro Edition (J2ME) is a JXTA version targeted to limited devices such as cellular phones. Using a relay server in a stationary network, even less capable devices running J2ME, can participate in a P2P network consisting of both mobile and desktop devices.

The application enables cellular phone users located anywhere in the world to share files through the JXTA network. Studies and tests shows that there are still too many constraints on cellular phones and in cellular environments to see any practical use of MOBster. However, the application has given us valuable information and revealed the main conditions that have to be fulfilled for such an application to become a success. Consequently we suggest a roadmap describing these conditions, and estimating when they can be met. We also suggested areas that need to be further explored for a Peer-to-Peer filesharing application on cellular phones to be a hit.

1 Introduksjon

Antall personer som har tilgang til Internett har økt enormt de siste årene. Den samme økningen ser vi nå også for brukere med mobiltelefon. Det er grunn til å tro at dette antallet vil fortsette å øke i årene som kommer, kanskje i en enda større hastighet, nå som vi står ovenfor utrullingen av tredje generasjons mobiltelefonsystem.

Mobiltelefonen har stor utbredelse, og er alltid med oss, på jobben, hjemme, på skolen og på bussen. Det faktum at mobiltelefoner er mobile og at mobiltelefoner etter hvert blir utstyrt med kamera, sensorer og posisjoneringssystemer som GPS (Global Positioning System) [55], betyr at mobiltelefoner i trådløse nettverk kan tilby et annet innhold enn hva stasjonære maskiner tradisjonelt kan tilby.

Pessimistiske beregninger viser at det til enhver tid finnes 10 Petabytes (10^{15} bytes) ledig lagringskapasitet og 10^8 MHz ledig prosessorkraft i Internett [4]. Peer-to-Peer (P2P) eller like-til-like arkitekturen søker å utnytte informasjonsmengden, båndbredden og prosessorkraften i Internett maksimalt. P2P teknologi gjør det mulig å dele ressurser og aksessere tjenester direkte mellom maskiner i et nettverk. P2P applikasjoner har flere egenskaper som gjør at de egner seg godt i trådløse omgivelser. De kan tilby høy tilgjengelighet, stor feiltoleranse og dynamisk oppdage andre terminaler og tjenester.

Fra slutten av 90-tallet og frem til i dag har fildelingsapplikasjoner som Napster [7], Gnutella [13] og KaZaA [60], vært de dominerende P2P-applikasjonene. Etterhvert som mobiltelefonene blir kraftigere, har det blitt mulig å realisere P2P-applikasjoner også på mobiltelefoner.

I denne oppgaven ønsker vi å se på mulighetene for å realisere en slik P2P-fildelingsapplikasjon på mobiltelefoner. Hva vil en overføring fra stasjonære til mobile omgivelser si for bruken av en P2P-fildelingsapplikasjon? Er mobile peer-er bare små begrensede utgaver av stasjonære peer-er?

1.1 Oppgaven

Målet med oppgaven var å *realisere P2P fildeling mellom mobiltelefoner*. Med *P2P fildeling* menes her at medlemmer eller *peer-er*[1] i et spontant anarkistisk nettverk skal kunne:

- gjøre egne filer tilgjengelige for andre
- søke etter filer blant andre peer-er
- laste ned filer fra andre peer-er

Med *filer* menes her tekst, bilder (.png, .jpg, .gif), audio- eller videoklipp. En *mobiltelefon* er her en peer i et spontant anarkistisk nettverk. Den benytter mobilnettet kun for å få konnektivitet til Internett.

Anarkistiske nettverk er kommunikasjonsnettverk uten noen sentralisert autoritet. Ulike anarkistiske nettverk basert på P2P har vokst frem de siste årene. Eksempler på dette er Napster, Gnutella og KaZaA. Spontane nettverk er selvorganiserende nettverk som oppstår for en bestemt anledning. De oppstår i forbindelse med, og varer så lenge som, enkelte mobile noder deltar i en kommunikasjonssesjon (mer om spontane nettverk i avsnitt 2.2).

Denne oppgaven skal forsøke å besvare en del problemstillinger som reises når P2P-fildeling flyttes fra stasjonære maskiner over på mobiltelefoner. Er det mulig å lage en brukbar fildelingsapplikasjon for en så begrenset terminal som en mobiltelefon? Hvilke plattformer eksisterer for å kunne utvikle P2P-fildelingsapplikasjoner? Hvordan vil taksering innvirke på bruken? Egner mobiltelefoner seg til å delta i fildeling?

[1] *Peer* er et innarbeidet uttrykk. *Peer* kommer til bli brukt i stedet for ”like” i resten av rapporten.

I forbindelse med oppstarten av prosjektet utarbeidet jeg en kravspesifikasjon (se vedlegg I) på bakgrunn av innspill fra deltakere i forskningsprogrammet. Kravspesifikasjonen viser hvilke intensjoner vi hadde med prosjektet.

I neste kapittel vil vi gå litt dypere inn i P2P-arkitekturen og forskjeller mellom Peer-to-Peer og klient/tjener. Kapittelet tar også for seg P2P-fildeling og hva som er spesielt med dette.

I kapittel 3, 4 og 5 ser vi nærmere på teknologiene som ligger til grunn for denne oppgaven. Først gir vi en oversikt over Java 2 Micro Edition (J2ME) og begrensninger som finnes i mobiltelefoner og deres omgivelser. Deretter gir vi et innblikk i P2P-plattformen JXTA, før vi går mer i dybden på JXTA for J2ME. I kapittel 6 presenteres prototypen som jeg utviklet i forbindelse med prosjektet. I kapittel 7 diskuteres resultatene fra utviklingsarbeidet og tester som ble gjort i etterkant. I kapittel 8 trekkes det konklusjoner ut fra resultatene som er kommet frem.

2 Bakgrunn

Internett har tre viktige bestanddeler som, slik situasjonen er i dag, ikke blir spesielt godt utnyttet. Disse tre er *informasjon*, *båndbredde* og *prosessorkraft*. En av årsakene til den dårlige utnyttelsen er den tradisjonelle klient/tjener tankegangen. Pessimistiske beregninger gjort for et par år siden sier at det til en hver tid finnes 10 PetaBytes (10^{15} bytes) ledig lagringsplass og 10^8 MHz ubrukt prosessorkraft i maskiner tilknyttet Internett [4]. Til tross for at det rulles ut mer og mer fiber, og at båndbredden i en fiber har økt med en faktor på 10^6 siden 1975 (en dobling hver 16. måned), opplever vi treghet inn mot de virkelig store nettstedene. Et eksempel på dette var nettaviser og spesielt CNN.com, som i perioder var nede 11. september 2001, rett etter at det første flyet dundret inn i World Trade Centre i New York. Når mange tusen maskiner forsøker å koble seg opp mot en eller flere sentraliserte tjenere samtidig, går det noen ganger galt. Det er mange flaskehalser. Internettforbindelsen, tjenerne og databasene har alle en grense for hvor mye trafikk de kan tåle.

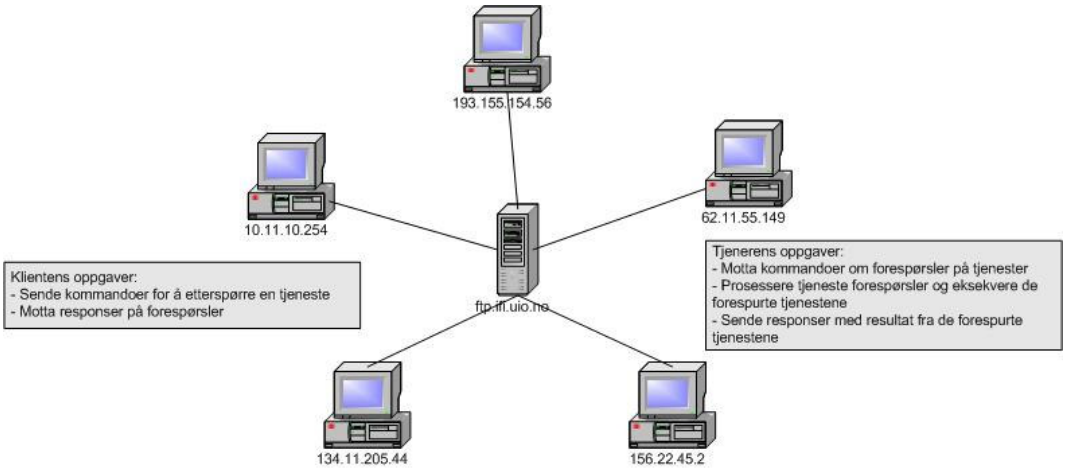
P2P søker å utnytte informasjonsmengden, båndbredden og prosessorkraften som finnes i dagens Internett maksimalt. Et P2P nettverk skiller seg fra vanlige klient/tjener nettverk ved at alle deltakere eller peer-er i nettverket kan kommunisere direkte med hverandre. Dette gjøres normalt uten noen hjelp fra sentraliserte tjenere eller ressurser. P2P arkitekturen kan brukes til å lage nettverk og tjenester med høy tilgjengelighet og feiltoleranse.

P2P-trafikk står for 60% av trafikken i Internett i følge en undersøkelse utført av Sandvine Inc. [30]. Dette har ført til at enkelte bredbåndsleverandører har valgt å legge om prisingen slik at kundene betaler for hvor mye trafikk de genererer i tillegg til en fast pris.

Dette kapittelet tar for seg de to viktigste arkitekturene i distribuerte systemer, klient/tjener og P2P. Først presenteres de to arkitekturene i avsnitt 2.1 og 2.2, før de sammenliknes i avsnitt 2.2.2. I avsnitt 2.2.3 og 2.2.4 ser vi på P2P fildeling og hvilke eksisterende applikasjoner som finnes.

2.1 Klient/ tjener

De fleste av tjenestene i dagens Internett er bygget med klient/tjener arkitektur. Eksempler på dette er World Wide Web (WWW), File Transfer Protocol (FTP) og e-post [4]. Klientene kobler seg opp mot tjeneren ved hjelp av en protokoll som for eksempel FTP, for å få tilgang til en ressurs (fil), se Figur 2-1. Tjeneren behandler forespørselen, og sender resultatet i retur til klienten. Klienten mottar resultatet.



Figur 2-1 Klient/tjener arkitektur

Tjenere kan også opptre som klienter ovenfor andre tjenere. Et eksempel på dette er Domain Name System (DNS). Når en klient forespør en DNS-tjener om en adresse, sjekker tjeneren først om den har IP-adressen til den forespurte maskinen. Har den det, sender den svaret i retur til klienten. Hvis den ikke har det, vil tjeneren opptre som en klient og sende forespørselen videre oppover i DNS-hierarkiet til en annen DNS-tjener. Litt avhengig av om det benyttes *rekursiv* eller *ikke-rekursiv* navigering[2], vil den opprinnelige DNS-tjeneren kontakte en eller flere tjenere for å finne svaret. Når DNS-tjeneren har funnet svaret, lagrer den resultatet i sin egen database, og sender resultatet til klienten.

2.2 Peer-to-Peer

P2P nettverk er i motsetning til klient/tjener nettverk, ikke avhengig av en sentralisert tjener for å kunne gi tilgang til tjenester. Peer-ene i et nettverk betraktes som likeverdige, og opptrer vekselvis som klienter og tjenere ovenfor hverandre[3] (se Figur 2-2). P2P-teknologi gjør det mulig å dele ressurser og aksessere tjenester direkte mellom maskiner i nettverket.

Sing Li [36] beskriver tre viktige egenskaper ved P2P-nettverk:

1) Høy tilgjengelighet

Datasystemer er ikke 100% pålitelige. Det kan skje feil med programvare, maskinvare og nettverkstilknytning eller brukerfeil. Det er flere måter å håndtere feil i distribuerte systemer på. Tre av disse er beskrevet her:

- Maskere feil – Meldinger kan retransmitteres hvis de ikke når fram til mottaker. Data kan bli lagret på flere maskiner slik at hvis en av dem feiler, kan andre overta.
- Tolerere feil – Systemer kan designes med tanke på å tolerere feil. Et eksempel er weblesere. Hvis det ikke oppnås kontakt med en webtjener, gir webleseren brukeren en feilmelding og lar brukeren selv bestemme hva han vil gjøre.
- Gjenoppretting etter feil – Systemet designes slik at tilstanden til persistente data kan gjenopprettes etter en krasj.

P2P-nettverk kan tilby høy tilgjenglighet av tjenester. En gruppe av peer-er kan gå sammen om å tilby en bestemt tjeneste. Om en eller to peer-er skulle bli midlertidig utilgjengelige, vil de gjenværende maskinene fortsatt kunne opprettholde tjenesten. P2P-nettverk har større toleranse ovenfor feil enn klient/tjener systemer. Fordi ressursene normalt har stor geografisk spredning, vil ikke feil på Internettforbindelser eller regionale strømutfall sette systemet ut av spill. Systemet er transparent med tanke på feil, det vil si at feilene som oppstår skjules for brukerne av tjenestene.

Den høye tilgjengeligheten og påliteligheten i P2P-systemer har gjort at det militære har begynt å vise interesse P2P-plattformer som JXTA (se kapittel 5 ”JXTA for J2ME™”).

2) Effektiv utnyttelse av ressurser

Som nevnt tidligere i dette kapittelet finnes det store mengder ubrukte ressurser i Internett. Privatpersoner kjøper kostbare maskiner med lagringskapasitet og prosessorkraft de sjelden kommer til å ha bruk for. Maskinene er dessuten sjelden i bruk hele tiden, noe som gjør at store mengder båndbredde står ubrukt til enhver tid.

[2] Ved rekursiv navigering vil DNS-tjeneren som blir etterspurt videreformidle forespørselen rekursivt hvis den ikke finner IP-adressen. Ved ikke-rekursiv navigering vil den forespurte DNS-tjeneren spørre en og en DNS-tjener om den har IP-adressen. Mer om navigering i DNS i [16].

[3] På engelsk kalles en maskin som vekselsvis opptrer som klient og tjener for *servent*, en sammenslåing av ordene *server* og *client*.

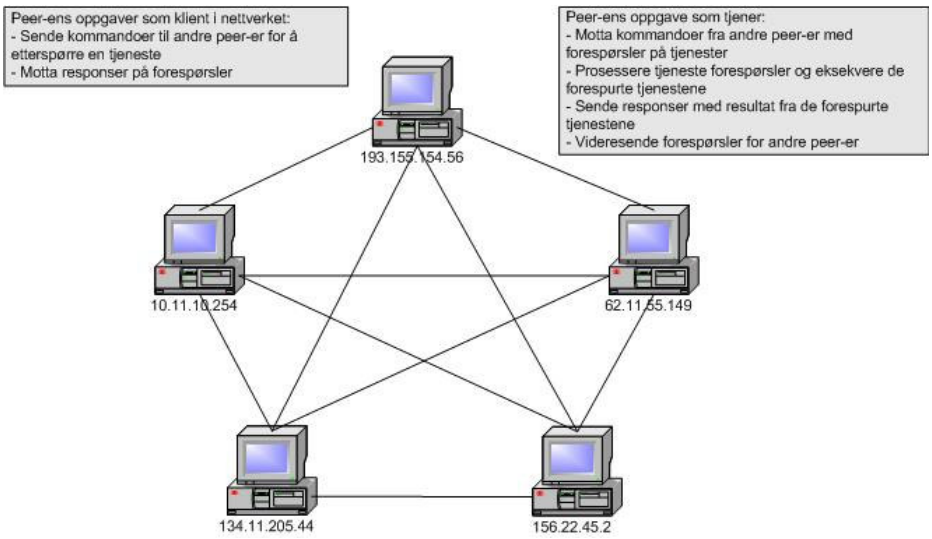
P2P arkitekturen utnytter ressursene i Internett på en mer effektiv måte. I stedet for at alle maskiner skal koble seg mot en sentralisert maskin, kan de heller kommunisere direkte, Peer-to-Peer. Dette gjør det mulig å utnytte den overkapasiteten som finnes, noe som kan føre til raskere responser, høyere tilgjengelighet og et mer variert tjenestetilbud.

3) "Ubegrenset" skalerbarhet.

Sentraliserte klient/tjener systemer har en grense for hvor mange klienter de kan betjene. Når belastningen blir for stor, kan hele systemet bli utilgjengelig. Klient/tjener systemer **skalerer** til en viss grense.

Et system er skalerbart hvis det forblir effektivt etter en signifikant økning i antall brukere og ressurser. Internett er et godt eksempel på et skalerbart system. I 1989 var det 130.000 datamaskiner koblet til Internett. I 1999 var det 56,2 millioner datamaskiner og 5,6 millioner webtjenere [16]. Det er flere måter å ta hensyn til skalerbarhet på. Det viktigste er å unngå flaskehalser i systemet. Forgjengeren til DNS (Domain Name System) var en stor fil som brukere kunne laste ned når de trengte å oversette en URL (Uniform Resource Locator) til en IP (Internet Protocol)-adresse. Dette fungerte fint når det kun var snakk om noen hundre maskiner som var koblet til nettet, men det ble snart klart at dette var en flaskehals både med tanke på ytelse og administrasjon. DNS fjernet flaskehalsen ved å dele navnetabellene mellom lokalt administrerte tjenere lokalisert rundt i Internett[4]. [16]

Med et godt design er det ved hjelp av P2P-arkitektur mulig å lage systemer som kan skalere til nesten det uendelige. Dette er ikke mulig med sentraliserte systemer. Se for deg en sentralisert fildelingsapplikasjon der brukere kan laste ned filer fra en og samme maskin. Det skal ikke mange brukere til før tjenesten oppleves som veldig treg og folk slutter å bruke den. For en P2P-fildelingsapplikasjon vil ikke dette være noe problem. Filene ligger spredt rundt på flere maskiner. Brukerne kobler seg opp direkte mot hverandre, og laster ned de filene de ønsker. Det finnes eksempler på P2P fildelingsapplikasjoner med mange millioner samtidige brukere (se Figur 2-6).



Figur 2-2 Peer-to-peer arkitektur.

[4] DNS er egentlig ikke et P2P-system, men et *hierarkisk* klient/tjener-system. Likevel er det med på å illustrere hvordan distribusjon av oppgaver og administrasjon kan være med på å øke skalerbarheten til et system.

Spontane nettverk
P2P nettverk blir ofte omtalt som spontane nettverk. Spontane nettverk må ikke forveksles med *Ad-hoc* nettverk. Ad-hoc er latinsk og direkte oversatt betyr det "for anledningen" eller "med dette for øyet". Ad-hoc nettverk er et fysisk nett, som oppstår for en bestemt anledning, som for eksempel mellom terminaler i et klasserom eller på en arbeidsplass. Et spontant nettverk er et "logisk" nettverk, med de samme egenskapene som Ad-hoc nettverk. Et spontant nettverk har nødvendigvis ikke noe å gjøre med geografisk nærhet, men kan oppstå mellom noder i et nettverk, uavhengig av hvor i verden nodene befinner seg. Et Ad-hoc nettverk har følgende egenskaper [14]:

- Ingen påkrevet infrastruktur. Nettet opprettes spontant, og er ikke avhengig av basestasjoner.
- Selvorganiserende: Nettverkstopologien i Ad-hoc nettverk rekonfigureres kontinuerlig.
- Feiltolerant: Hvis en node mister kontakten med nettverket, rekonfigureres nettet og alternative forbindelser settes opp.

For mer informasjon om Ad-hoc nettverk, se [14]. I denne oppgaven skal vi konsentrere oss om P2P-fildeling for mobiltelefoner i et spontant nettverk. Telefonene trenger ikke nødvendigvis å være i nærheten av hverandre for å kunne kommunisere.

2.2.1 Hvorfor P2P?

Vi har vært inne på litt av fordelene med P2P-systemer i avsnittet over. På grunn av sine egenskaper åpner P2P-arkitekturen for en variasjon av tjenester. P2P er kanskje mest kjent for fildelingsapplikasjoner som Napster, Gnutella og KaZaA. Selv om P2P har fått en del negativ oppmerksomhet de siste årene på grunn av illegal distribuering av filer med mønsterbeskyttelse, har P2P mer å tilby enn bare enkel aksess til stjålne musikk- og videofiler. P2P er mer enn søk og fildeling.

P2P applikasjoner kan deles inn i tre hovedkategorier: IM (Instant Messaging), fildeling og distribuert prosessering. IM-tjenester som ICQ, AOL og MSN Messenger lar brukere sende meldinger direkte til hverandre i sanntid. SETI@home (Search for Extraterrestrial Intelligence) [4] er et eksempel på hvordan P2P kan brukes i distribuert prosessering. SETI@home var en distribuert skjermsparer-basert applikasjon, som gjorde beregninger på data fra et radioteleskop, og dermed bidro til søket etter utenomjordisk liv. Mer enn 3 millioner brukere deltok i prosjektet. I fremtiden er det ventet at det vil utvikle seg markedsplasser der kunder kan kjøpe billig prosesseringskraft [4]. P2P-fildeling vil vi komme nærmere inn på i avsnitt 2.2.3.

Hvorfor P2P på mobiltelefon?
Mens P2P i kablede nettverk har eksistert en tid og vært jobbet mye med, har det blitt gjort lite forskning når det gjelder P2P i trådløse nettverk på begrensede terminaler og spesielt mobiltelefoner [15]. Med begrenset terminal mener vi:

- Fra 160 kB til 512 kB tilgjengelig minne.
- 16-bit eller 32-bit prosessorer.
- Lite strømforbruk. Benytter som regel batterier som strømforsyning.
- Konnektivitet gjennom en variabel forbindelse, ofte trådløs, med begrenset båndbredde.

De aller fleste mobiltelefoner passer inn under denne beskrivelsen. Mobiltelefonen har stor utbredelse, og er alltid med oss, på jobben, hjemme, på skolen og på bussen. Det faktum at mobiltelefoner er mobile og at mobiltelefoner etter hvert blir utstyrt med kamera, sensorer, grensesnitt for Ad-hoc kommunikasjon (se avsnitt 2.2.2) og posisjoneringssystemer som GPS (Global Positioning System), betyr at P2P på mobiltelefoner i trådløse nettverk kan tilby et annet innhold enn hva stasjonære maskiner tradisjonelt kan tilby. Dette er innhold som har begrenset "holdbarhetsdato" og som kanskje bare har lokal interesse. Et eksempel er temperaturen i rommet en melding er sendt i fra. Temperaturen kan være interessant hvis den som mottar meldingen vurderer å bevege seg til den andre brukeren. Hvis beskjeden som mottas er gammel, er det derimot ikke sikkert at temperaturen er så viktig. Nytten av innholdet varierer med tid og rom.

De fleste nye mobiltelefoner blir levert med GPRS (General Packet Radio Service). GPRS er i motsetning til GSM en pakkesvitsjet løsning for å sende data. Med GPRS er mobiltelefonen alltid tilkoblet nettverket, og klar for å ta i mot og sende trafikk. Dette gjør mobiltelefonen bedre egnet til til å delta i P2P-nettverk enn tidligere.

P2P fildeling for mobiltelefoner
Med P2P fildeling for mobiltelefoner mener vi at fildelingen skjer på mobiltelefonen, i den forstand at filene som deles ligger lagret på mobiltelefonen. Alle filer lastes direkte opp fra eller ned til mobiltelefonen. Mobiltelefonen opptrer som en peer eller like i forhold til de terminalene som telefonen deler filer med.

Mobiltelefoner og stasjonære PC-er brukes i forskjellige kontekster. En stasjonær PC brukes enten hjemme, på skole eller på jobb. En mobiltelefon er med over alt. Når en bruker sender SMS (Short Message Service) eller spiller et spill på en mobiltelefon, er oppmerksomheten mobiltelefonen får fra brukeren begrenset. Når en bruker sitter foran en PC, kan brukeren som regel gi all sin oppmerksomhet til PC-en. En mobiltelefon brukes gjerne i sammenhenger og situasjoner der brukerens oppmerksomhet er opptatt med andre oppgaver som å kjøre bil, gå, vente i kø eller snakke med andre mennesker [14].

Som nevnt tidligere eksisterer det flere P2P-fildelingsapplikasjoner for stasjonære maskiner. Vi ønsker å se på mulighetene for å realisere en slik applikasjon på mobiltelefoner. Hva vil en overføring fra stasjonære til mobile omgivelser si for bruken av en P2P-fildelingsapplikasjon? Er mobile peer-er bare små begrensede utgaver av stasjonære peer-er? Disse spørsmålene diskuteres nærmere i kapittel 7.

I neste avsnitt skal vi se på fordeler og ulemper med både P2P-arkitekturen og den tradisjonelle klient/tjener-arkitekturen. Er P2P det beste valget for en fildelingsapplikasjon på mobiltelefoner?

2.2.2 Klient/ tjener vs. Peer-to-Peer

Klient/tjener er per i dag den mest utbredte av disse to arkitekturene. I begynnelsen var Internett et rent P2P-nettverk, der alle maskinene kunne koble seg direkte opp til hverandre ved hjelp av IP-adresser. Etterhvert har Internett utviklet seg, og fått en mer og mer sentralisert arkitektur. Det er flere årsaker til dette. Blant annet har mangelen på IP-adresser gjort det nødvendig å tildele IP-adresser dynamisk[5] til PC-er som kobler seg på Internett. Dette gjør det vanskelig å kommunisere direkte mellom to maskiner. De siste årene har vi sett en endring tilbake mot det opprinnelige desentraliserte Internett, i og med fremveksten av P2P-applikasjoner. Mange snakker også om ”the Semantic Web”, det semantiske Internett, som neste generasjons Internett. *Semantic Web* kommer til endre Internetts infrastruktur fra et sentralisert system til Peer-to-Peer, noe som muliggjør sanntids søk og øyeblikkelig tilgang til ressurser på Internett [62].

Klient/tjener-arkitektur krever mindre prosessering og ressurser på klienten enn på tjeneren. Dette er fordelaktig i systemer der noen terminaler er mer ressurssvake enn andre. Tjeneren behandler forespørsler på vegne av klienten, og sender resultatene i retur til klienten. Mobiltelefoner er begrensede i forhold til PC-er. Det at klient/tjener krever så lite ressurser hos klienten taler for å bruke en slik arkitektur for programmer på mobiltelefoner. Siden tjeneren skal behandle forespørsler fra mange klienter, kreves det desto mer av en tjener, både når det gjelder båndbredde, minne og prosesseringskraft.

[5] IP-adresser tildeles dynamisk fra en DHCP (Dynamic Host Configuration Protocol) tjener

P2P-nettverk er ikke avhengige av sentraliserte tjenere for å gi tilgang til tjenester i nettet. P2P lar peer-er oppdage hverandre dynamisk og kommunisere med hverandre enten direkte eller via andre peer-er. P2P egner seg godt i omgivelser der terminaler kobler seg til og fra hele tiden, som i mobile nettverk. P2P lar peer-er danne grupper som tilbyr tjenester. Ansvaret for tjenesten deles mellom peer-ene i gruppen. Skulle noen peer-er forsvinne, vil de gjenværende peer-ene kunne opprettholde tjenesten. P2P skalerer godt, fordi trafikk og prosessering fordeles på alle peer-er i en gruppe eller et nettverk.

Det er mulig å lage klient/tjener systemer som er robuste i forhold til feil og stor pågang. Slike systemer er kostbare å utvikle og vedlikeholde. Dublerte aksesser, replikering av tjenestene og maskinvare og programvare koster penger. For å tilby enda større tilgjengelighet og feiltoleranse kan det ofte være nødvendig å plassere utstyr i forskjellige lokaler, noe som er med på å øke kostnadene ytterligere. P2P kan ofte tilby samme grad av robusthet ved å spre nettverks- og ressurskrav ut i P2P nettverket. P2P utnytter potensialet og ressursene som allerede finnes i Internett på en bedre måte enn klient/tjener arkitekturen.

P2P på sin side lider av uforutsigbarhet. Det er ikke mulig å vite hvor lenge en enkelt peer er tilgjengelig. Hvis peer-en er den eneste som tilbyr en tjeneste, er det da heller ikke mulig å vite hvor lenge en enkelt tjeneste er tilgjengelig [4]. Ikke alle tjenester egner seg like godt i dynamiske omgivelser. Det er ikke sikkert at P2P-arkitekturen ville vært den beste for en nettbank. Ikke alle terminaler egner seg like godt til å delta i P2P-nettverk. Som navnet Peer-to-Peer sier, må alle terminaler kunne opptre som likeverdige i et P2P-nettverk.

Det er ikke mulig å si at en av disse to arkitekturene passer best i enhver situasjon. I en situasjon kan klient/tjener være den beste, i en annen er P2P best. For noen systemer er det best å bruke en kombinasjon av klient/tjener og P2P. Det er en avgjørelse utviklerne må ta for hvert enkelt system.

Hvorfor P2P og ikke klient/tjener på mobiltelefonen?
Som for alle andre type terminaler er det fordeler og ulemper med å benytte P2P-arkitektur for programmer på mobiltelefoner. Siden mobiltelefonen er en begrenset terminal, taler mye for å bruke en klient/tjener-arkitektur. En mer ressurssterk terminal som for eksempel en PC, kan ta seg av de tyngste beregningene. På en annen side befinner mobiltelefoner seg i mobile omgivelser, hvor dekningen kan variere og forbindelser brytes, og P2P egner seg godt for slike dynamiske omgivelser.

Forskningsprogrammet Anarkistiske Nettsamfunn ved Telenor FoU jobber med spontane nettverk og P2P. Siden oppgaven gjøres i samarbeid med dette programmet, vil fokuset i denne oppgaven være på P2P-fildeling for mobiltelefoner. I denne oppgaven ønsker vi å undersøke om dette egner seg på mobiltelefoner. Valg av P2P-arkitekturen var gitt av forskningsprogrammet, men implementasjonen benytter faktisk en kombinasjon av klient/tjener og P2P arkitekturen (se kapittel 6 ” Prototypen”).

I neste avsnitt skal vi se nærmere på fildeling i P2P-systemer.

2.2.3 Fildeling i P2P-systemer

P2P-fildeling tok for alvor av med Napster [7] mot slutten av 90-tallet. Siden da har P2P omtrent vært synonymt med fildeling. Med P2P-fildeling kan brukere eller peer-er som deltar i et nettverk selv velge hvilke filer som skal gjøres tilgjengelige for andre. Brukeren kan søke etter filer hos andre peer-er og laste disse ned. Det er ingen sentralisert ressurs som styrer hvilke filer som er tilgjengelige eller hvem som kan laste ned hva. Til sammen utgjør alle peer-ene i nettverket en tjeneste, som alle som er med i nettverket kan benytte. Ved at hver enkelt peer bidrar med sine filer sikres det et mangfold i tjenesten.

Tiden det tar før en nylig publisert Internettside dukker opp på en søkemotor som *Google* er noen uker. Av og til hender det at treffene fra søk er utdatert, og at linken til nettstedet ikke lenger finnes. Med P2P-teknologi er det mulig å lage søkemotorer som kan finne den aller nyeste og mest oppdaterte informasjonen [17].

Hvorfor er P2P fildeling viktig?
Tidligere i kapittelet har vi snakket mye om det tekniske aspektet ved P2P. P2P kan gi høy tilgjengelighet, det skalerer godt og gir en effektiv utnyttelse av ressurser. P2P og da spesielt P2P fildeling har også et bruksmessig og sosialt aspekt. Bidraget fra hver enkelt bruker i systemet er med på å forme en tjeneste. Når brukerne bruker systemet er de samtidig med på å tilby filer til andre peer-er. Det er en *vinn-vinn* situasjon der alle parter som deltar tjener på samarbeidet. Det er ingen personer eller firmaer som tjener penger på det som foregår, unntatt de som leverer nettverksforbindelser. Med P2P står brukere fritt til å utvikle egne applikasjoner og systemer som fungerer uten sentralisert styring og på tvers av nettverk. Kontrollen skifter fra organisasjoner til individer [9].

Hvorfor er P2P fildeling så populært?
Det er nok flere faktorer som har gjort at P2P fildeling er så populært som det er. For det første har utbredelsen av bredbånd[6] gjort det mulig for brukere å være tilkoblet Internett til en fast pris hver måned. Dette gjør det mulig for disse brukerne å dele sine filer med andre, uten at det påfører dem noen ekstrakostnader. Båndbredden som tilbys er mange ganger større enn på analoge modemforbindelser, noe som gjør at det går raskere å laste ned filer. Raskere nedlastning og større tilgjengelighet av innhold er viktig for brukere av en fildelingstjeneste.

En annen årsak til at P2P fildeling er blitt så populært er MP3[7]-formatet. MP3 formatet er et komprimeringssystem for musikk. Formatet reduserer størrelsen til en sang, uten at det går nevneverdig ut over kvaliteten på lyden. En sang i CD-kvalitet kan komprimeres med en faktor på mellom 10 og 14 ganger uten merkbar forskjell på kvalitet. MP3 gjør det mulig å laste ned en musikkfil i løpet av minutter istedenfor timer. Med MP3 ble det mer interessant å dele informasjon direkte mellom brukere. Brukere kunne laste ned all den musikken de ønsket seg. Napster var en P2P-fildelingsapplikasjon beregnet på MP3-formatet (se avsnitt 2.2.4). Napster var for mange det første møtet med P2P fildeling, og åpnet opp en helt ny verden.

En tredje årsak til populariteten til P2P fildeling er mangfoldet i informasjon. I dag er det nesten ikke grenser for hva du kan få tak i via KaZaA eller Gnutella. Omtrent alt som lar seg digitalisere er tilgjengelig for nedlastning. Du kan finne bilder, musikk, filmer, programvare og mye mer. Mye av innholdet som ligger tilgjengelig for nedlastning er imidlertid mønsterbeskyttet. Mange forbinder derfor fildeling med ulovlig distribuering av filer.

Tragedy of the commons
Etterhvert som tjenestene utvikler seg er det mange som velger å ikke bidra med informasjon, men bare ta del i godene. I enkelte samfunn som deler ressurser, kan det oppstå noe som kalles "Tragedy of the Commons". *Tragedy of the Commons* vil si et overforbruk av felles ressurser slik at hele systemet bryter sammen [4]. I enkelte P2P-systemer er det mulig å bruke andres ressurser (båndbredde og lagringskapasitet) uten å måtte bidra med noe selv. Dette er med på å redusere nytteverdien av nettverket. Etterhvert som flere og flere peer-er velger å ikke dele sine ressurser med andre, vil de peer-ene som fortsatt deler sine ressurser bli overbelastet. Nettverket vil bevege seg fra et P2P-nettverk mot den klassiske klient/tjener arkitekturen. Til slutt vil nettverket kollapse, noe som ødelegger for alle.

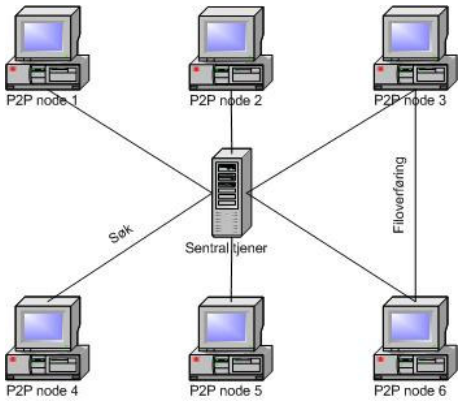
[6] Ifølge SSB defineres bredbånd som Internettforbindelser på over 384 kbit/s
[7] MPEG er en forkortelse for Moving Picture Experts Group. Denne gruppen har utviklet et komprimeringssystem for video og data. MPEG inneholder et system for å komprimere lyd, som kalles MPEG audio Layer 3. Dette er det vi kjenner som MP3.

2.2.4 Eksisterende P2P-fildelingsapplikasjoner

Det finnes flere forskjellige P2P-fildelingsapplikasjoner tilgjengelig for PC-er. De tre mest kjente er Napster [7], Gnutella [13] og KaZaA [60]. P2P fildelingsapplikasjoner deles gjerne inn i to kategorier, avhengig av graden av sentralisering i systemet. Et system som behandler alle peer-er likt og ikke har noen sentralisert søkefunksjonalitet kalles en **ren** P2P løsning. Systemer som bruker spesielle sentraliserte enheter (som en indekstjener til indeksering av filer) kalles et **hybrid** P2P system.

Sandvine Inc. [30] deler P2P-fildelingsapplikasjoner inn i tre kategorier: **ren** P2P, **hybrid** P2P og en blanding av disse. Sandvine viser med dette hvordan forskjellige generasjoner av P2P-fildelingsapplikasjoner har utviklet seg fra en sentralisert modell, over til den helt desentraliserte og nå har landet på en mellomting av disse.

Napster (Sentralisert)
Første generasjon P2P fildeling, som for eksempel Napster, var et eksempel på et **hybrid** P2P-system (se Figur 2-3). I Napster lastes navnet på alle filer som en bruker deler, opp i en katalog på en sentralisert tjener (indekstjener). Dette skjer hver gang brukeren logger seg på nettverket. Alle søk foretas mot denne indekstjeneren, mens overføringen av filer skjer direkte mellom peer-er.



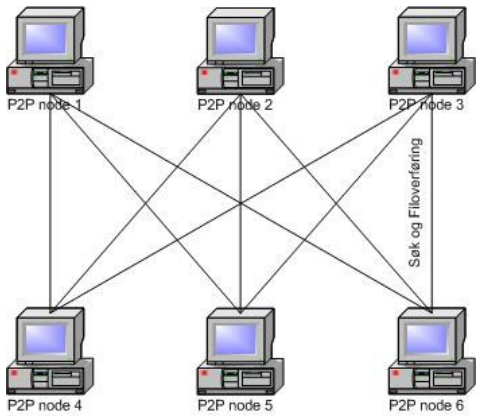
Figur 2-3 Sentralisert P2P-nettverk

Sentraliserte tilnærmelser som Napster gir raske søk/forespørsler, gitt at det finnes god nok kapasitet mot den sentraliserte tjeneren [30]. Feiltoleransen og skalerbarheten til et slikt system er begrenset, fordi alle operasjoner er avhengige av å ha forbindelse med den sentraliserte tjeneren [7]. Det er enklere å stenge en tjeneste som Napster, fordi alle som bruker tjenesten er avhengig av kontakten med indekstjeneren. I rene P2P systemer vil det være vanskeligere for myndigheter å stenge en fildelingstjeneste.

Gnutella (Desentralisert)
Andre generasjons P2P fildelingsapplikasjoner, som for eksempel Gnutella[8], er **rene** P2P-system som implementerer en distribuert modell uten noen sentralisert tjener eller node (se Figur 2-4). Alle søk blir kringkastet til alle peer-er ved hjelp av en multi-hopp "oversvømmelse" algoritme som sender dem fra terminal til terminal. Filene lastes ned direkte mellom to peer-er.

[8] Dette er slik Gnutella var i starten i følge [13]. Gnutella har endret seg etterhvert, og likner nå mer på tredje generasjons P2P-systemer, se Figur 2-5 [2].

Desentraliserte systemer som Gnutella tilbyr en høy grad av feiltoleranse fordi systemets operasjoner ikke er avhengig av en dedikert maskin. Likevel krever algoritmen at en maskin har minst en forbindelse med en annen peer. Et problem med Gnutella er at denne "oversvømmingen" av nettverket med forespørsler er med på å begrense Gnutellas skalerbarhet. Blir nettet for stort vil maskinene tilknyttet nettet til slutt ikke gjøre annet enn å behandle forespørsler. [13]

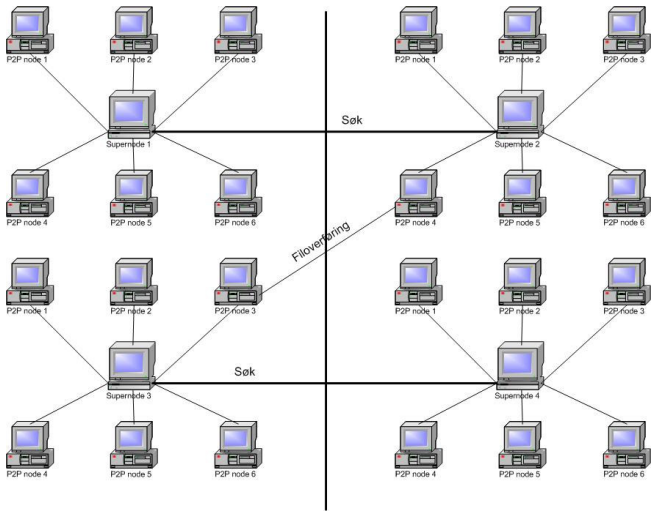


Figur 2-4 Desentralisert P2P-nettverk

For å kunne begrense omfanget av søk har Gnutella et TTL (Time To Live)-felt i alle pakker som inneholder forespørsler. Dette feltet minker med èn for hver node søket er innom. Når verdien av feltet er null, kastes pakken. TTL-feltet har to fordeler. For det første begrenser dette søk og hindrer at nettverk oversvømmes av forespørsler. For det andre er det med på å hindre at pakker kan sveve rundt i nettverket i lengre tid uten mål og mening. Verdien til TTL-feltet er i utgangspunktet satt til syv. I praksis vil dette si at et søk begrenses til rundt 10.000 peer-er [13]. Gnutella omtaler dette som en **horisont**.

I følge [17] er det bare 30% av de som bruker Gnutella som bidrar med å dele sine egne filer. 70% er gratispassasjerer. I rene P2P løsninger er det vanskeligere å "straffe" de som ikke bidrar med filer til fellesskapet.

KaZaA (Kontrollert desentralisert)
Tredje generasjons P2P fildeling, som for eksempel KaZaA, er en slags blanding av en **ren** og en **hybrid** P2P-arkitektur. KaZaA benytter noe som de kaller *supernoder* eller *ultrapeers* for å effektivisere søk (se Figur 2-5). Hvis en bruker sitter på en kraftig PC med en rask Internettforbindelse, kan denne maskinen bli utpekt som en *supernode*. Supernoden inneholder en liste med noen av de filene som er tilgjengelig i KaZaA-nettverket, og hvor disse kan finnes. Når en peer foretar et søk, søkes det først på den supernoden som er nærmest. Denne supernoden sender søkene videre til andre supernoder, som også behandler søket. Brukeren laster deretter ned den ønskede filen direkte fra peer-en som har den på sin maskin. [60]



Figur 2-5 Kontrollert desentralisert P2P-nettverk

Med en blanding av en sentralisert og desentralisert modell kan KaZaA tilby god skalering ved å redusere antallet noder som er involvert i behandling av søk, og dermed også redusere trafikken i nettet. Søkene blir effektive fordi få noder er involvert i søket. Det gir rask respons, og samtidig rimelig oppdatert informasjon. KaZaA er dessuten mer tolerant ovenfor feil enn sentraliserte modeller.

KaZaA har også metoder for å gi brukere som bidrar med mye informasjon fordeler. Hvis to brukere forsøker å laste ned samme fil, vil den som bidrar med mest i nettverket få førsteretten. Dette er med på å hindre gratispassasjerer og scenarier som *Tragedy of the Commons*.

Figur 2-6 *Statusbar* hentet fra KaZaA mars 2003. Viser at det er over 3,2 millioner brukere online som deler 690 millioner filer.

2.3 Oppsummering

I dette kapittelet har vi sett på de to mest utbredte arkitekturene som brukes i distribuerte systemer, klient/tjener og P2P. Vi har sett på egenskaper ved begge arkitekturene, og hvordan de skiller seg fra hverandre. P2P tilbyr høy tilgjengelighet, effektiv utnyttelse av ressurser og god skalerbarhet. Klient/tjener er den mest utbredte arkitekturen. Med klient/tjener stilles det mindre krav til ytelse og ressurser på klienten. Deretter så vi på forskjellen mellom *Ad-hoc* nettverk og spontane nettverk.

I avsnitt 2.2.3 så vi nærmere på fildeling i P2P nettverk. Vi så på mulige årsaker til at P2P-fildeling er så populært. Etter dette beskrev vi tre forskjellige P2P-fildellingsapplikasjoner for stasjonære maskiner, Napster, KaZaA og Gnutella. Fildelingsapplikasjoner deles i to kategorier, **rene** og **hybride**. Rene systemer benytter ingen sentralisering, mens hybride systemer har sentraliserte indekstjenere som peer-ene foretar søk i mot.

I denne oppgaven ønsker vi å utforske potensialet i P2P for begrensede terminaler. Valg av P2P-arkitekturen var gitt av forskningsprogrammet Anarkistiske Nettsamfunn, men implementasjonen benytter faktisk en kombinasjon av klient/tjener og P2P arkitekturen. Dette kommer vi nærmere inn på i kapittel 6 ”Prototypen”.

I de neste fire kapitlene skal vi se på prototypen og utviklingen av denne. Kapittel 3, 4 og 5 tar for seg teknologiene som er brukt i forbindelse med utviklingen. Kapittel 6 beskriver selve prototypen.

3 J2ME – Java™ 2 Micro Edition

Java™ er et moderne objektorientert språk, med innebygget sikkerhet og med støtte for nettverksoperasjoner. Java kildekode blir ikke kompilert til maskinkode, men til bytekode som kjører på en Java Virtual Machine (JVM). JVM-en gjør de nødvendige oversettelsene mellom bytekode og maskinkode. På denne måten blir Java uavhengig av underliggende operativsystem.

Java har gradvis utviklet seg fra Java 1 Standard Edition til Java 2. I juni 1999 redefinerte Sun hele arkitekturen til Java-plattformen, og delte den inn i Standard Edition (SE), Enterprise Edition (EE) og Micro Edition (ME). SE er laget med tanke på standard applikasjoner for PC-er og arbeidsstasjoner. EE er laget for tjenere i et nettverk og inneholder all den funksjonalitet som behøves i slike stormaskiner. ME er beregnet for små begrensede terminaler. Dette kapittelet handler om Java 2 Micro Edition, som forkortes J2ME og uttales "*J to me*", se også [38] og [57].

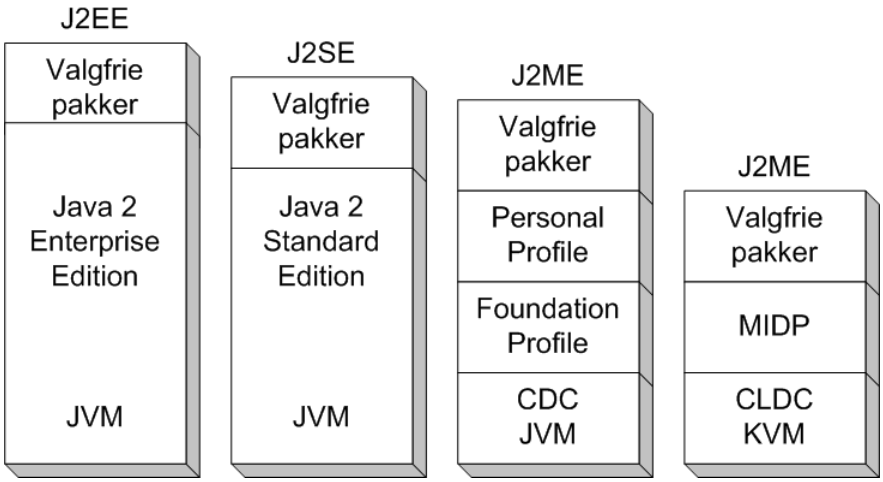
Markedet for mobile terminaler med nettverkstilknytning har hatt en kraftig økning de siste årene. Spesielt mobiltelefoner har stor utbredelse. Mobiltelefonene blir stadig kraftigere og får flere og flere funksjoner. J2ME er Sun Microsystems Java-versjon for forbrukerelektronikk. Eksempler på dette er mobiltelefoner, PDA-er (Personal Digital Assistant), set-top-boxer[9], CD-spillere, videospillere og smartkort.

I en undersøkelse gjengitt på Micro Java Network [37] og på [40], hevdes det at Java vil være den dominerende teknologien på mobiltelefoner i årene fremover. I 2002 var det 50 millioner mobiltelefoner med støtte for Java. I år 2007 vil 450 millioner mobiltelefoner ha støtte for Java. Det vil utgjøre 74% av alle terminalene som selges det året.

Som navnet tilsier er J2ME en liten kjøretidsomgivelse. Kjøretidsomgivelsen tar seg av oversettelse mellom bytekode og maskinkode i tillegg til at den tilbyr et sett med standard API-er (Application Programming Interface). J2ME deles inn i **konfigurasjoner**, som består av virtual machine (VM)[10], kjerne-biblioteker, klasser og API-er og **profiler**. På det nåværende tidspunkt finnes det to konfigurasjoner: Connected Limited Device Configuration (CLDC) og Connected Device Configuration (CDC). En profil er en utvidelse av en konfigurasjon, og tar for seg spesielle behov for en gruppe terminaler, for eksempel mobiltelefoner. Et eksempel på et spesielt behov for mobiltelefoner, kan være et eget brukergrensesnitt-API som er tilpasset de små skjermene på mobiltelefonene. Profilen garanterer interoperabilitet mellom terminaler innen samme gruppe. [28]

En konfigurasjon definerer en Java plattform for en **horisontal** kategori eller gruppe av terminaler med omtrent samme krav til minne og prosessorkraft. Både PDA-er og videospillere havner innen samme horisontale kategori. Profiler tar i motsetning til konfigurasjoner for seg terminaler i en **vertikal** kategori. Videospillere fra forskjellige leverandører vil havne innen samme vertikale kategori.

[9] Set-top-box er en komponent som gir brukere tilgang til digitale-sendinger på TV-en. Settes opp på TV-en, derav navnet.
[10] Virtual Machine oversettes med **virtuell maskin** i resten av oppgaven



Figur 3-1 Oversikt over Java-arkitekturen

I dette kapittelet skal vi se nærmere på J2ME og hvordan det kan brukes under utvikling av applikasjoner for mobiltelefoner. Vi skal først se litt nærmere på de forskjellige konfigurasjonene og profilene som finnes i J2ME, spesielt CLDC (avsnitt 3.1.1) og MIDP 1.0 (avsnitt 3.3). Deretter skal vi se på neste generasjon av MIDP og hvilke muligheter versjon 2.0 vil gi. Avsnitt 3.7 tar for seg begrensningene som finnes i dagens mobiltelefoner og deres omgivelser. I avsnittet "Ytelsesdrevet programmering" ser vi nærmere på hvordan applikasjonsutviklere kan effektivisere sine programmer.

3.1 Konfigurasjoner

Foreløpig eksisterer det to J2ME konfigurasjoner (se Figur 3-1). Disse har to forskjellige versjoner av den virtuelle maskinen. En konfigurasjon inneholder:

- støtte for Java programmeringsspråk
- Java Virtual Machine (JVM)
- grunnleggende Java bibliotek og API-er

I de to neste avsnittene presenteres de to konfigurasjonene.

3.1.1 CLDC - Connected Limited Device Configuration

CLDC er designet for terminaler med begrenset prosessorkraft og minne. Den definerer et minimum av hva som kreves av en Java plattform. CLDC er laget for terminaler som passer til følgende karakteristikker:

- Fra 160 kB til 512 kB tilgjengelig minne
- 16-bit eller 32-bit prosessorer
- Lite strømforbruk (bruker som regel batterier som strømforsyning)
- Konnektivitet gjennom en variabel forbindelse, ofte trådløs, med begrenset båndbredde

CLDC bruker en JVM som heter KVM. "K" indikerer at den virtuelle maskinen er liten. Dette er i følge [28] den minste komplette virtuelle maskin som lages. CLDC er oppover-kompatibel. Derfor er det mulig å kjøre J2ME applikasjoner i en J2SE omgivelse.

Både J2SE og J2EE inneholder et mangfold av biblioteker som er til nytte for utviklere. Bibliotekene krever mye minne. Siden minne er svært begrenset på CLDC-terminaler, tilbyr CLDC kun et minimum av bibliotek. Konnektivitet er viktig for begrensede terminaler. CLDC har derfor noen ekstra bibliotek nettopp for å støtte kommunikasjon til og fra terminalen.

I denne oppgaven kommer vil til å fokusere på CLDC, da det er denne konfigurasjonen som brukes på dagens mobiltelefoner.

3.1.2 CDC - Connected Device Configuration

CDC er designet for neste generasjons mobile terminaler, som har mer minne og prosessorkraft tilgjengelig enn CLDC-terminaler. Typiske CDC-terminaler har følgende egenskaper:

- 2 MB eller mer minne tilgjengelig for Java plattformen
- 32-bit prosessor
- Høy båndbredde på nettverksforbindelse, som regel TCP/IP

CDC vil ikke bli diskutert mer i denne rapporten. Mer informasjon om CDC finnes på [57].

3.2 Profiler

Profiler tar i motsetning til konfigurasjoner for seg terminaler i en **vertikal** kategori. De er beregnet til et markedssegment, en gruppe av terminaler eller innenfor en type applikasjoner. Forbrukerelektronikk spenner over et stort spekter av terminaler og utstyr. De forskjellige terminalene har forskjellige krav og funksjonaliteter. Et eksempel på dette er at en komfyr neppe vil ha like mye glede av støtte for e-post som en mobiltelefon ville ha. Under er en liste over forskjellige profiler:

- MIDP - Mobile Information Device Profile. Mobiltelefoner og PDA-er.
- FP - Foundation Profile. Kombineres med andre profiler over CDC (se Figur 3-1).
- PP - Personal Profile. Full støtte for GUI (Graphical User Interface) og Applet.
- PBP - Personal Basis Profile. TV, CD-spillere og videospillere.

3.3 MIDP – Mobile Information Device Profile

MIDP spesifikasjonen definerer arkitekturen og API-ene som behøves til tredje parts programutvikling av applikasjoner for mobiltelefoner. Alle mobiltelefoner har noen felles egenskaper, slik som begrenset minne, liten skjerm og begrenset båndbredde. Det er versjon 1.0 av MIDP som brukes i dagens utgave av J2ME. MIDP 2.0 spesifikasjonen er allerede klar, og blir å finne i mobiltelefoner i løpet av 2003. MIDP 2.0 omtales i avsnitt 3.6. Fokuset i denne rapporten vil ligge på MIDP 1.0 fordi det er denne versjonen som brukes i dagens mobiltelefoner. MIDP 1.0 vil heretter bli omtalt som MIDP.

De viktigste funksjonene MIDP tilbyr er [38]:

- Brukergrensesnitt gjennom *javax.microedition.lcdui*[11]
- Nettverkstilkobling gjennom *javax.microedition.io*
- Persistent datalagring med RMS (Record Management System) gjennom *javax.microedition.rms*

Sammen med CLDC tilbyr MIDP et fullverdig utviklingsmiljø for mobile applikasjoner.

[11] Java sitt bibliotek er delt opp i pakker (*package*). Pakkene deles opp i kategorier og underkategorier, som skilles med "."(punktum). For mer informasjon henviser vi til [57].

3.3.1 MIDlet

Applikasjoner som kjører på enheter som støtter MIDP kalles *MIDlets*. Akkurat som **applet**-er kontrolleres MIDlet-er av programvaren de kjøres på [61]. En MIDlet må arve MIDlet-klassen, for at programvaren som kjører MIDlet-en skal ha mulighet til å kontrollere den. MIDlet-klassen inneholder API-er for å kalle opp, stoppe, starte på nytt og terminere MIDlet-er. Programvaren som håndterer MIDlet-ene kan administrere flere MIDlet-er som kjører på den samme virtuelle maskinen. MIDlet-ene blir da pakket sammen i en *MIDlet-suite,* som kan dele ressurser seg i mellom.

MIDlet-ene i en MIDlet-suite, blir lagret i en .jar-fil[12]. I forbindelse med generering av .jar-filen, vil det også bli laget en beskrivelses-fil, .jad (Java Application Descriptor), som beskriver innholdet i .jar-filen. Eksempel på informasjon i en .jad-fil kan være versjon, størrelse og forfatter (se Eksempel 3-1).

```
MIDlet-Name:            MOBster
MIDlet-Version:         1.2.1
MIDlet-Vendor:          Telenor FoU
MIDlet-1:               MOBster
MIDlet-2:               Chat
MIDlet-Jar-URL:         mobster.jar
MIDlet-Jar-Size:        27500
MIDlet-Description:     P2P-filesharing application
```

Eksempel 3-1 .jad-fil (Java Application Descriptor)

MIDP sørger for portabilitet mellom forskjellige mobiltelefoner slik at samme MIDlet kan brukes på en Nokia telefon så vel som en Siemens telefon. MIDP implementasjonen innkapsler forskjeller mellom enheter ved å ta hånd om skjermstørrelser og tilordning av knapper. Dessverre er det ikke alltid slik at MIDlet-er som fungerer på en mobiltelefon, automatisk fungerer på en telefon fra en annen produsent. Terminalprodusentene kan ha forskjellige implementasjoner av CLDC og MIDP.

Trådløse Java applikasjoner deles inn i to kategorier [31]:

- **Lokale applikasjoner** (også kalt enkeltstående programmer). Gjør alle operasjoner på terminalen, og trenger ikke aksess til eksterne data via trådløst nettverk. Et eksempel på slike lokale applikasjoner er kalkulatorer og enkle spill.
- **Nettverksapplikasjoner.** Består av komponenter som kjører både på terminalen og på andre maskiner i nettverket. Er avhengig av aksess til trådløst nettverk.

3.3.2 MIDlet vs. Applet

En Applet er et Java-program som kan inkluderes i en HTML-side (HyperText Markup Language), se Eksempel 3-2. Dette gjøres omtrent på samme måte som med et bilde. Det benyttes en spesiell HTML-tag eller etikett som ser slik ut:

```
<APPLET CODE="HelloWorld.class" WIDTH=150 HEIGHT=25>
</APPLET>
```

Eksempel 3-2 Applet som inkluderes i en HTML-side. `HelloWorld.class` **er navnet på Applet-en.**

[12] *Java Archive* fil-format. .jar kan komprimere og pakke sammen flere filer.

Når du bruker en nettleser til å se på en side som inneholder en applet, lastes koden til applet-en ned til din maskin og eksekveres i nettleserens JVM [50]. Livssyklusen for en applet ser slik ut:

- Lastet inn – en instans av applet-en har blitt laget, men applet-en er ikke initialisert.
- Stoppet – applet-en er initialisert men inaktiv
- Startet – applet-en er aktiv
- Ødelagt – applet-en har terminert og instansen er klar for å bli plukket opp av søppeltømmingen

På tilsvarende måte er MIDlet et Java-program som kan lastes ned eller overføres til en mobiltelefon. Koden til MIDlet-en lastes ned og kjøres på mobiltelefonens JVM eller KVM. Det er mange likheter mellom applet-er og MIDlet-er [11]. I stedet for en nettleser, er det en AMS (Application Management Software) innebygget i terminalen som tar seg av administrasjonen av MIDlet-er. En stor fordel med ekstern administrasjon av MIDlet-er er at applikasjonen kan overstyres ved innkommende anrop til mobiltelefonen eller når det kommer SMS (Short Message Service). Under er en oversikt over livssyklusen til en MIDlet:

- Pause - en instans er laget, men er fortsatt inaktiv
- Aktiv - MIDlet-en er aktiv
- Ødelagt - MIDlet-en har terminert, og er klar for å bli plukket opp av søppeltømmingen

Mer om likheter mellom applet-er og MIDlet-er kan leses i [11].

3.3.3 RMS – Record Management Store

For enkelte applikasjoner er det viktig å kunne lagre data persistent, slik at programmet kan få tilgang til dataene ved en senere anledning, selv om programmet avsluttes i mellomtiden. MIDP tilbyr en slik mekanisme gjennom RMS (Record Management Store). RMS er en enkel database som kan inneholde data som forblir persistente etter at MIDlet-en avsluttes. Når brukeren starter MIDlet-en på nytt får programmet tilgang til de samme dataene igjen. Databasefilene (*record stores*) er plattformavhengige. Det vil si at hver enkelt terminalprodusent står fritt til å implementere databasefilen på sin måte. Filene kan ikke flyttes mellom mobiltelefoner.

RMS har et API som tilbyr følgende funksjonalitet:

- Lar MIDlet-er legge til og fjerne poster (*record*) i en databasefil.
- Lar MIDlet-er i samme *suite* dele poster i en databasefil (kan aksessere hverandres poster direkte).
- Hindrer MIDlet-er i andre *suiter* i å få tak i en databasefil. Dette er av sikkerhetsmessige årsaker.

For mer informasjon om RMS, se [57].

3.4 Sikkerhet

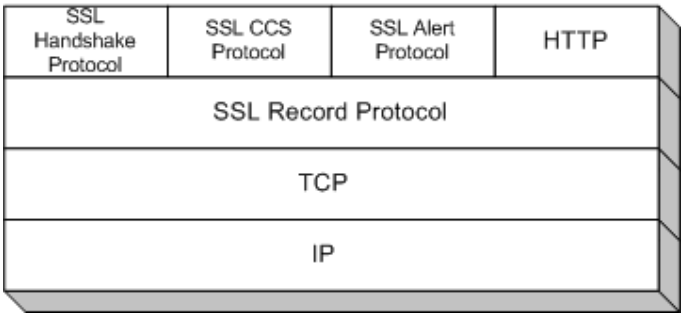
All informasjon som overføres over trådløse nettverk er et mulig mål for avlytning. Dataene oppholder seg ikke bare i trådløse forbindelser, men går som regel mot en sentralisert tjener. På veien dit kan dataene passere mange nettverk. Sensitiv informasjon kan fanges opp og misbrukes. Eksempel på sensitiv informasjon som kan utnyttes er kredittkortnummer, pin-koder og annen personlig informasjon.

For å tilby en sikker forbindelse må både klient og tjener implementere en sikkerhetsmodell. En måte å beskytte sensitiv informasjon på er gjennom kryptering. Senderen krypterer dataene før de sendes over en trådløs link, hvorpå den autoriserte tjeneren tar i mot de krypterte dataene og dekrypterer slik at innholdet i meldingen blir forståelig. SSL (Secure Socket Layer) er en protokoll som er utbredt i applikasjoner for PC-er (se Figur 3-2) og som tilbyr en sikker forbindelse mellom to terminaler. SSL er for stor og krevende til å kunne kjøre på begrensede terminaler som en mobiltelefon. En miniversjon av SSL har derfor blitt utviklet, nemlig *kSSL* se 3.4.1. [32]

HTTPS er en sikker utgave av HTTP. HTTPS fås ved å kjøre HTTP over SSL (se Figur 3-2). MIDP 1.0 spesifikasjonen sier at plattformen **kan** ha støtte for HTTPS, men at dette ikke er noe krav. Dette innebærer at terminalprodusentene står fritt til å velge om støtte for HTTPS skal implementeres. Dette fører til at mobiltelefoner kan ha forskjellig J2ME-funksjonalitet, noe som vanskeliggjør utvikling av programmer for mobiltelefoner. I versjon 2.0 av MIDP **må** HTTPS implementeres.

3.4.1 SSL / kSSL

SSL (Secure Socket Layer) lar klienter (nettlesere) og webtjenere kommunisere over en sikker forbindelse. Den tilbyr kryptering, autentisering og dataintegritet for å sikre informasjon som utveksles over usikre offentlige nettverk. SSL er i dag godt utbredt i e-handelsapplikasjoner for PC-er. Mange nettsteder, som for eksempel nettbanker, benytter seg av sikker HTTP. Etterhvert som flere og flere applikasjoner dukker opp på mobiltelefoner, øker behovet for SSL også her. SSL er som nevnt lite egnet på mobiltelefoner. En veldig komplisert håndtrykk-prosess samtidig med hyppig utskiftning og utveksling av nøkler gjør SSL lite egnet for mobiltelefoner. Sun har derfor utviklet en forenklet versjon av SSL, kalt "kilobyte" SSL (kSSL).



Figur 3-2 SSL protokoll stakk

kSSL

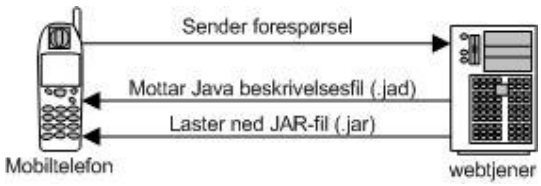
kSSL er en klient-side implementasjon av SSL, som støtter de mest vanlige *chiffer*[13]. Det vil si at mobiltelefoner som implementerer kSSL kan kommunisere med tjenere som har støtte for SSL uten at utviklerne behøver å endre på noe hos tjeneren. Ved å gjenbruke nøkler og sesjoner samt å begrense antallet chiffer som det er støtte for, har Sun Microsystems klart å lage en liten og effektiv utgave av SSL. kSSL leveres nå med J2ME Wireless Toolkit (omtalt i avsnitt 3.5). [32]

3.5 Utvikling av MIDlet-er

Ved utvikling av applikasjoner for mobiltelefoner kan det være nyttig å bruke et verktøy som for eksempel J2ME Wireless Toolkit (J2WTK). J2WTK inneholder funksjoner for å kompilere Java-filer og pakke disse i .jad-filer og .jar-filer. I tillegg følger det med seks emulatorer som kan gi et inntrykk av hvordan applikasjonen blir seende ut på mobiltelefonen. J2WTK har også funksjoner for å overvåke minneforbruk og nettverkstrafikk til og fra emulatorene. Dette er nyttig, siden minne og båndbredde er begrenset på mobiltelefoner.

Etter at MIDlet-en er utviklet er det på tide å teste den på en mobiltelefon. Det er to måter å overføre MIDlet-en til en mobiltelefon på, enten via seriellkabel eller ved nedlastning fra en webtjener (se Figur 3-3). Dette kalles OTA (Over-The-Air), og har fått navnet fordi mobiltelefonen setter opp en trådløs forbindelse over Internett til webtjeneren som MIDlet-en ligger lagret på.

[13] *Chiffer* eller *chiffer suite* vil si en samling av krypteringsteknologier. I kSSL sitt tilfelle er det RSA_RC4_128_MD5 og RSA_RC4_40_MD5. Mer om dette i [42].



Figur 3-3 OTA - nedlastning fra webtjener.

3.6 MIDP 2.0

Versjon 1.0 av MIDP tilbyr flere API-er for utvikling av applikasjoner (MIDlet-er). MIDP 1.0 har blant annet API-er for brukergrensesnitt, utgående HTTP og persistent lagring. Versjon 1.0 har begrensninger som gjør det vanskelig å utvikle en del type applikasjoner. Blant annet mangler den støtte for sikkerhet, mulighet for å bruke *sockets* og støtte for RGB-grafikk[14]. MIDP 2.0 er laget for å ta hånd om noen av disse begrensningene.

20. november 2002 var endelig versjon av MIDP 2.0 spesifikasjonen klar. En betaversjon av J2ME Wireless Toolkit (versjon 2.0), som implementerer MIDP 2.0, er tilgjengelig fra Sun sine websider [57]. Selv om spesifikasjonen er klar, vil det ta noe tid før mobiltelefoner med MIDP 2.0 vil bli tilgjengelig i stort volum. Foreløpige estimater tyder på at de første MIDP 2.0 terminalene vil bli tilgjengelige på markedet sommeren 2003 [19].

Avsnittene 3.6.1 til 3.6.5 tar for seg de viktigste forskjellene mellom versjon 1.0 og versjon 2.0.

3.6.1 Sikkerhet

I avsnitt 3.4 snakket vi om hvor viktig sikkerhet er. Selv om MIDP 1.0 støtter HTTPS, kan ikke utviklere stole på at den er tilgjengelig, fordi det ikke er noe krav i spesifikasjonen at HTTPS **må** implementeres. Den eneste nettverksprotokollen mobiltelefoner med MIDP 1.0 trenger å ha støtte for er HTTP. I MIDP 2.0 kreves også HTTPS. Det er kSSL som benyttes i MIDP 2.0.

MIDP inkluderer støtte for HTTPS gjennom CLDC Generic Connection Framework (GCF) i pakken *javax.microedition.io*. Å sette opp en HTTPS forbindelse gjøres ved å spesifisere URL-en og deretter åpne forbindelsen med Connector.open (se Eksempel 3-3)

```
String URL = "https://www.centrebet.com";
HttpsConnection hc = null;
hc = (HttpsConnection)Connector.open(url);
```

Eksempel 3-3 Sette opp en HTTPS-forbindelse

3.6.2 Multimedia

Noe av det mest spennende med MIDP 2.0 er multimedia API-ene. Med de nye API-ene har utviklere mulighet til å få mobiltelefonen til å spille de lydene de ønsker fra programmene de lager. Mange av dagens mobiltelefoner, blant annet Nokia, gir brukeren mulighet til å komponere melodier selv. Med Multimedia API-et får utviklerne tilgang til å lage og spille av musikk i MIDlet-er. 3. generasjons mobilnett, UMTS (Universal Mobile Telecommunications System) har kostet nettoperatørene store summer. Dette må tjenes inn på trafikk og tjenester. Multimedia i form av bilde, lyd og video vil være viktig med tanke på generering av trafikk og mulige tjenester.

[14] RGB-bilder blir representert med en *int* for hver piksel. 8 bit for opasitet (gjennomsiktighet), rød, grønn og blå verdier.

Spesifikasjonen krever at MIDP 2.0 **må** kunne spille av WAV (Waveform audio file)-filer, men terminalprodusentene er fri til å implementere støtte for andre lydformater som for eksempel .MP3.

3.6.3 Spill API

MIDP 2.0 har fått et eget API beregnet spesielt på spillutviklere. API-et gir mulighet for bedre grafikk på mobiltelefonene. Spill API-et lar utviklere dele skjermen inn i flere **lag**, og bevege lagene i forhold til hverandre. Et lag kan inneholde et bakgrunnsbilde, mens et annet lag viser et romskip. MIDP 2.0 har også støtte for RGB-grafikk noe som gir utviklerne mulighet til å manipulere bilder. MIDP 1.0 har ingen støtte for å dele skjermen inn i flere lag og svært begrensede muligheter for grafikk noe som gjør det vanskelig å utvikle spill.

Med Java har det blitt mulig å bytte ut spillene som ligger lagret på telefonen til fordel for nye spill som lastes ned via Internett. Foreløpig eksisterer det få andre typer programmer en kan laste ned og bruke på sin Java-telefon. Spill derimot, finnes på en rekke nettsteder som for eksempel http://www.jippii.no og http://www.eurobate.com. Høyst sannsynlig er det gjennom spill at brukere får mest utbytte av Java på mobiltelefonen sin i første omgang.

3.6.4 Tillatelse og signering av kode

MIDlet-er har tilgang og mulighet til å sette opp nettverksforbindelser over GSM eller GPRS. Datatrafikk koster penger, og MIDlet-er som setter opp nettverksforbindelser kan dermed være med på å påføre brukeren av programmet uønskede kostnader [19]. MIDlet-er som setter opp forbindelser ut fra din mobiltelefon kan også være med på å utgjøre en sikkerhetsrisiko. MIDlet-en kan sende fra seg sensitiv informasjon om deg og din bruk av programmet. I lys av dette er det i MIDP 2.0 innført et konsept som kalles **tillatelser** (*permissions*). Brukere har muligheten til å klassifisere programmer som **tiltrodd** og **ikke tiltrodd**. Kode som ikke er tiltrodd, må få tillatelse fra brukeren før den kan sette opp en forbindelse. Kode kan bli klassifisert som tiltrodd hvis de som har utviklet programvaren signerer koden digitalt, og brukerens terminal kan verifisere signaturen.

3.6.5 Andre nye muligheter

MIDP 2.0 har mange nye interessante funksjoner, som gir helt nye muligheter med tanke på applikasjonsutvikling på mobiltelefoner. I tillegg til de vi har nevnt i avsnittene over kan vi også nevne:

- Forbedrede muligheter for brukergrensesnittet. Enklere å plassere elementer som input-felt, tekst og bilder, på skjermen. Bedre algoritmer for utseende og enklere håndtering av input.
- Støtte for å sette opp *sockets* og å sende datagram. I følge spesifikasjonen er det valgfritt å implementere dette i MIDP 2.0. Med sockets har en mulighet til å sette opp direkte kommunikasjon mellom to mobiltelefoner, og utveksle data begge veier. Terminalprodusentene står fritt til å implementere *sockets*.
- *Push Registry*. Gjør det mulig å starte opp MIDlet-er som en respons på innkommende nettverksforbindelser. Det er mange mulige anvendelser av en slik funksjon. En mulighet kan være å bruke det til å gi beskjeder til bekjente (en billig variant av SMS). En annen mulighet er å tilby *Web Services* (omtalt i avsnitt 7.2) fra mobiltelefonen.

I neste avsnitt skal vi se på begrensninger som finnes i mobiltelefoner og deretter gå igjennom noen tips til hvordan utviklere kan gjøre applikasjonene sine mest mulig effektive.

3.7 Utvikling av applikasjoner for mobiltelefoner

Programutviklere som designer applikasjoner som skal kjøre på store kraftige maskiner, er vant til at de har tilnærmet ubegrenset minne og prosesseringskraft til rådighet. I tillegg har maskinene som oftest stabile nettverksforbindelser på minimum 100 Mbit/s. Applikasjonene skal gjerne presenteres på skjermer som er 17” eller større.

Disse utviklerne vil møte mange utfordringer når de skal jobbe med de begrensede ressursene som er tilgjengelige i trådløse mobile terminaler som mobiltelefoner. Skjermstørrelse, batterikapasitet, minne, prosesseringskraft og input-muligheter må tas hensyn til. Utviklere møter også andre utfordringer i omgivelsene til applikasjonene: Mobilitet og trådløse nettverk, som vanligvis tilbyr en lavere båndbredde og er mindre pålitelige enn kablede nettverk.

Dagens forbrukere har blitt vant til datamaskiner og programvare med nesten ubegrenset tilgang på ressurser. De er vant til fleksible og pålitelige systemer. Brukere ønsker terminaler med applikasjoner som er enkle, intuitive og pålitelige.[12]

For å kunne utvikle gode applikasjoner på mobile terminaler kreves det at utviklerne klarer å sette seg inn i de spesielle utfordringene som finnes for denne typen terminaler. I tillegg må de kunne dra nytte av de fordelene mobile terminaler har.

I avsnitt 3.7.1 ser vi nærmere på begrensningene som finnes i dagens mobiltelefoner. I avsnitt 3.7.2 ser vi på trådløse omgivelser, og hva som kjennetegner disse. Vi ser på båndbredde, forsinkelse og kostnader. Avsnitt 3.7.3 gir tips til hvordan utviklere kan lage effektive programmer til mobiltelefoner.

3.7.1 Begrensninger med mobiltelefoner

De siste årene har vi opplevd en enorm vekst i antallet trådløse terminaler, mest for mobiltelefoner. I følge en undersøkelse gjort av Norsk Gallup, gjengitt i VG den 16.01.2003 [47], har 87% av alle nordmenn over 15 år en mobiltelefon. Mobiltelefonene får mer og mer prosessorkraft og lagringskapasitet og bedre skjermer. Veksten i antall mobiltelefoner har ført til en økt innsats i arbeidet med å tilpasse mye av den teknologien som er utviklet for stasjonære maskiner til håndholdte terminaler [34]. Denne overgangen er ikke alltid like enkel. Alle begrensingene i håndholdte mobile terminaler, spesielt mobiltelefoner, gjør at utviklere må tenke helt nytt med tanke på hvordan de bruker minne, presenterer informasjon og tar i mot input fra bruker.

En del prosjekter har slått feil, og ikke blitt så store som forventet. Et eksempel på dette er WAP (Wireless Application Protocol). Etter at finansmarkedene hausset det mobile Internettet (WAP) opp i skyene i 2000, viste det seg å bli en gedigen flopp da det ble sluppet på markedet [44]. I 2001, ett års tid etter lanseringen av WAP, brukte 3% av Norges befolkning WAP jevnlig. Overdrevne forventninger om "Internet in your hand," trege mobilnett, dårlige brukergrensesnitt og mangel på interessante tjenester er noen av forklaringene.

I avsnittene under skal vi gå mer inn på de begrensningene som finnes for utvikling av applikasjoner på mobiltelefoner (fra en utviklers synspunkt):

Minne
Hvor mye minne som er tilgjengelig varierer fra telefon til telefon. Nokias business-telefon, 6610, har 725 kB tilgjengelig minne til hele telefonen. Kun 64 kB er tilgjengelig for hver enkelt Java-applikasjon (maksimum 6 applikasjoner) [54].

Prosessorkraft
Hastigheten på prosessorer varierer veldig fra telefon til telefon. Som regel er det snakk om 16-bits prosessorer. Nokia 9210 kommer med en 32-bits RISC-prosessor. Nokia sier ikke noe om hvor mange MHz det er snakk om. I følge [59] er det normalt med rundt 20MHz på mobiltelefoner.

Input til telefonen
De fleste mobiltelefoner tilbyr som regel et tastatur med 12 knapper: de 10 numeriske knappene pluss stjerne (*) og skigard (#). I tillegg kommer meny-knappene. Sony Ericsson har lansert en mobiltelefon, modell P800, som har et display som er følsomt for berøring. Brukere kan gi input til telefonen ved hjelp av en blyant-liknende peker. Selv om det fortsatt er stor forskjell mellom mobiltelefoner og PDA blir skillet mindre og mindre.

Skjerm
Lenge har skjermer på mobiltelefoner vært meget begrenset. Normalt har det vært snakk om 96x54 piksler med 1 bit dybde (sort/hvit). I den siste tiden har det skjedd store fremskritt på denne fronten. Nokia 6610, som det er referert til tidligere, har en skjerm på 128x128 piksler med 4096 farger (12 bit). Nyere telefoner fra Sony Ericsson skal ha hele 65536 farger. Det er likevel stor forskjell mellom mobiltelefoner og datamaskiner som gjerne har millioner av farger og en mye større oppløsning.

Figur 3-4 Nokia 3510i med fargeskjerm. Brukt som test-telefon i prosjektet.

Batterikapasitet
Batterikapasiteten varierer fra telefon til telefon og etter hvordan telefonen brukes. Batterikapasitet måles som regel i **taletid**, det vil si tiden mobiltelefonen er i aktiv bruk enten til tale eller datatrafikk, og **standby-tid**, som er tiden mobiltelefonen er på, men ikke brukes aktivt. På de første GSM-telefonene var standby-tid typisk 10 timer, mens taletid var 1 time. På Nokia 6610 er standby-tiden 320 timer og taletiden 2-6 timer.

Selv om det har vært en betydelig utvikling når det gjelder minne, prosessor og skjerm de siste årene, kan ikke batteriene vise til den samme fremgangen. Batteriene har riktignok blitt en god del mindre rent fysisk, og lavt forbruk av strøm i mobiltelefonene har sørget for god standby-tid. Taletiden har det derimot vært lite endring på de siste årene. Det er derfor viktig for utviklere å lage applikasjoner som holder strømforbruket så lavt som mulig.

3.7.2 Trådløse omgivelser
I forrige avsnitt tok vi for oss begrensningene som eksisterer i dagens mobiltelefoner. I dette avsnittet tar vi for oss mobiltelefonenes trådløse omgivelser. Trådløse omgivelser har en del egenskaper som må tas hensyn til under utvikling av applikasjoner for mobiltelefoner [12].

Trådløse nettverk er i stor grad mer upålitelige enn kablede nettverk.
Når det gjelder GSM er dekningen i Norge rimelig god. Det er likevel fortsatt mange steder der det er vanskelig å få dekning. Eksempler på dette er kjellere, tunneler, betongbygg, på fjellet og i de dype skoger. Mobiliteten til mobiltelefoner øker faren for at en forbindelse brytes eller blir forringet. Kapasiteten i mobiltelefonnettverk kan i sentrale og tett bebodde strøk være for liten til å ta unna toppene i trafikken. Dette kan føre til sperr for de som forsøker å gjøre nye anrop. I følge [34] oppleves det mer feil i trådløse nettverk enn i kablede nettverk.

Mobiltelefoner er dyre i bruk.
Foreløpig er det dyrt å bruke mobiltelefoner til å sende datatrafikk. Båndbredden som tilbys er lav i forhold til kablede nettverk og WLAN (Wireless Local Area Network) – områder. For tiden er det to måter å sende datatrafikk til og fra mobiltelefonen på: GSM-data, som er en linjesvitsjet forbindelse eller GPRS (General Packet Radio Service) som er pakkesvitsjet. HSCSD (High Speed Circuit Switched Data) er en variant av linjesvitsjet GSM-forbindelse som tar i bruk flere kanaler, og dermed kan tilby en høyere overføringskapasitet. Oversikt over hastigheter i de forskjellige nettverkene finnes i Tabell 3-3 (side 36).

Som nevnt er datatrafikk dyrt på mobiltelefoner. For GSM-data betaler brukeren for hvor lenge han er oppkoblet. Med HSCSD betaler han i tillegg for hver kanal som benyttes. Med GPRS betaler brukeren for mengden trafikk som sendes til og fra mobiltelefonen. I motsetning til linjesvitsjede forbindelser hvor det tar lang tid å sette opp en forbindelse, er GPRS alltid påkoblet. Tabell 3-1 og Tabell 3-2 viser en oversikt over priser på datatrafikk.

Linjesvitsjet data

	Etablering	Månedspris	Pris pr. minutt
GSM-data (Telenor)	100,-	15,-	0,89 (primær)
GSM-data (Netcom)	0,-	0,-	0,89
HSCSD (Telenor)	100,-	15,-	1,88 (uansett)
HSCSD (Netcom)	0,-	0,-	0,89 (pr. kanal)

Tabell 3-1 Priser på linjesvitsjet dataoverføring (per 12. mars 2003)

Pakkesvitsjet data

	Opp til 0,5 MB	0,5-20 MB	Over 20 MB
GPRS (Telenor) Kr. per kilobyte	0,10	0,015	0,01

	Første 20MB	over 20MB (per kilobyte)
GPRS (Netcom storbruker)	200	0,015

Tabell 3-2 Priser på pakkesvitsjet dataoverføring (per 12.mars 2003)

Det er viktig at utviklere er klar over egenskapene til de trådløse omgivelsene som mobiltelefonene befinner seg i. Det oppleves mer feil i trådløse nettverk enn i kablede nettverk og datatrafikk i mobiltelefonnettverk er kostbar. Utviklere må derfor prøve å begrense datatrafikken til det som er ytterst nødvendig. Programmene må også lages med tanke på at Internettforbindelsen kan brytes vilkårlig.

Båndbredde og forsinkelse
Hastigheten på et nettverk måles i båndbredde og forsinkelse. Båndbredde er definert som mengden data som overføres over en forbindelse per tidsenhet. Måleenheten er bits per sekund. Forsinkelse er tiden det tar for ett element med data å bevege seg fra avsender til mottaker over et nettverk. Måleenhet er sekunder. Det er vanlig å bruke *round trip time* som et mål på forsinkelse. *Round trip* er tiden det tar fra en melding sendes fra avsender, overføres til en mottaker, behandles hos mottaker, overføres til avsender og mottas hos avsender. Vi bruker verdien for *round trip* i Tabell 3-3. [12]

For store mengder data er som regel båndbredde viktigst fordi tiden det tar å laste ned data avhenger av båndbredden. Forsinkelse har her ikke så mye å si, fordi den utgjør så lite av det totale tidsforbruket. For mindre mengder data, er det forsinkelse som er viktigst. Både båndbredde og forsinkelse avhenger av hvilken nettverksteknologi som brukes. Tabell 3-3 viser en oversikt over forsinkelse og båndbredde i GSM og GPRS.

	Båndbredde (kilobits per sekund)	Typisk første HTTP round trip forsinkelse	Typisk påfølgende HTTP round trip forsinkelse
GSM – data	9,6 kbps	5-10 sekunder	2-3 sekunder
GSM High-Speed-Circuit-Switched Data (HSCSD)	opp til 43 kbps	5-10 sekunder	2-3 sekunder
GPRS	5-50 kbps	5-8 sekunder (kan være mer når nettverket er overbelastet)	2-4 sekunder (kan være mer når nettverket er overbelastet)

Tabell 3-3 Båndbredde og forsinkelse i forskjellige nettverk [12]

Fordi forsinkelsen er så stor i mobiltelefonnettverk, egner ikke mobiltelefoner seg for alle type applikasjoner. Ta for eksempel et online-spill, der deltakerne skal skyte på hverandre med romskip. Med en forsinkelse på to til tre sekunder vil det være vanskelig å fastslå hvem som skjøt hvem først.

Den store forsinkelsen og de store kostnadene forbundet med datatrafikk fører til at applikasjonsutviklere bør prøve å begrense antall HTTP-kall [12]. De bør heller prøve å samle opp informasjon som skal sendes, og sende mer informasjon i samme HTTP-kall, hvis dette er mulig. Det er også viktig å gjøre mengden informasjon som overføres så liten som mulig.

3.7.3 Ytelsesdrevet programmering[15]

Ytelsesdrevet programmering er et konsept som presenteres i artikkelen om "Wireless Software Design Techniques" [34]. På begrensede terminaler er det viktig å få applikasjonene til å gå så raskt som mulig. Artikkelen tar for seg noen tips til hvordan utviklere kan optimalisere ytelsen til applikasjonene de lager:

- **Initialiser aldri et objekt til null.** Initialiseringen tas hånd om automatisk av KVM-en.
- **Bruk lokale variabler i stedet for klassevariable så ofte som mulig.** Aksessen til lokale variabler er mye raskere.
- **Færrest mulige metodekall.** KVM-en laster opp og lagrer en "tilstand" hver gang det gjøres et metodekall. Under er et eksempel på hvordan antall metodekall kan begrenses:

  ```
  for(int i=0; i < obj.length; i++) {
     //gjør noe med tabell-elementet
  }
  ```

  Over blir lengden på tabellen kalkulert hver gang programmet går igjennom *for*-løkken. Det er mer effektivt å definere en lokal variabel, og kun kalle metoden en gang.

  ```
  int lengde = obj.length;
  for(int i=0; i < lengde; i++) {
     //gjør noe med tabell-elementet
  }
  ```

- **Færrest mulig objekter.** Lag objekter som kan resirkuleres. Aktivering av objekter fører til at den virtuelle maskinen en gang må deaktivere og kaste disse. Dette reduserer ytelsen. I stedet for å returnere objekter i metodekall, send heller en referanse til objektet som returverdi og endre objektets verdier i stedet. Dette er ikke helt i tråd med god objektorientert tankegang, men det øker ytelsen. Ved å gjenbruke objekter bruker programmet mindre minne og prosessoren bruker ikke like mye tid på søppeltømming.

[15] Ytelsesdrevet programmering, fritt oversatt fra det engelske ordet *Performance-driven Programming*

- **Unngå å sette sammen objekter med '+' operatoren**. Dette fører til opprettelse av objekter med påfølgende søppeltømming, noe som bruker både minne og prosessorkraft. Det er mer effektivt å bruke et `StringBuffer` [57].
- **Unngå synkronisering**. Hvis en metode bruker lang tid på å kjøre, bør utvikleren forsøke å plassere metode-kallet i en egen tråd.

Ytelse og brukeropplevelse
Brukeren bryr seg ikke om hvor rask applikasjonen er, bare den oppleves som rask [12]. Dette betyr at det ikke nødvendigvis gjør noe at brukeren må vente 10 sekunder på et svar, så lenge brukeren selv opplever dette som raskt. Det er flere måter å gjøre dette på. En måte er å hele tiden informere brukeren om hva som skjer. Et eksempel på dette er å vise tilstandsrapporter til brukeren som ”Beskjeden er sendt” eller ”Venter på svar fra nettverket”. Brukeren slipper da å lure på om beskjeden er sendt, eller om noe gikk galt når han skulle taste på tastaturet. En annen mulighet er å bruke timeglass eller tellere for å vise hvor lenge brukeren har ventet. Brukeren opplever fremdrift og at noe skjer.

Konklusjon
Det er viktig å ta hensyn til begrensningene som finnes i mobile terminaler. Fokuset bør likevel settes på alle mulighetene som finnes i mobiltelefoner og de mulighetene som åpner seg når dette kombineres med Java 2 Micro Edition. En mobiltelefon kan bli et ”vindu” inn mot nettverk der andre maskiner står for prosessering. På den måten kan operasjoner som før bare kunne gjøres fra stasjonære maskiner, gjøres fra alle steder med GSM-dekning. Mobiltelefonen kan være med på å øke tilgjengeligheten til tjenester.

Det er viktig med et enkelt og funksjonelt brukergrensesnitt som legger opp til minst mulig skriving for brukeren. Brukeren må få god informasjon om hva som skjer.

3.8 Oppsummering
Dette kapittelet har tatt for seg J2ME (Java 2 Micro Edition) og utvikling av applikasjoner for mobiltelefon. Først så vi på de forskjellige konfigurasjonene og profilene som finnes i J2ME og spesielt CLDC og MIDP. Deretter så vi på sikkerhet og hvordan utviklere kan bygge sikkerhet inn i applikasjoner på mobiltelefoner. I avsnitt 3.6 så vi på neste generasjon av MIDP og hvilke nye muligheter versjon 2.0 kommer til å gi når det gjelder sikkerhet, grafikk og kommunikasjon. Til slutt ga kapittelet en oversikt over de begrensingene som finnes i dagens mobiltelefoner og mobilnettverk. I avsnittet "Ytelsesdrevet programmering" ble det gitt noen tips til hvordan applikasjonsutviklere kan effektivisere sine programmer.

Med Java på mobiltelefoner har utviklere tilgang på et programmeringsspråk som er kraftig nok til å lage P2P-applikasjoner for mobiltelefoner. Neste kapittel tar for seg et rammeverk, JXTA, for utvikling av slike.

4 JXTA

De fleste av dagens P2P-applikasjoner overlapper hverandres funksjonalitet på enkelte områder. Microsofts MSN Messenger tilbyr IM (Instant Messaging) og fildeling. KaZaA [60] tilbyr hovedsakelig fildeling, men også IM. Alle P2P-applikasjoner har en del grunnleggende funksjoner for å kunne kommunisere med omverdenen. Peer-er må kunne oppdage hverandre, søke etter informasjon og utveksle informasjon.

Dessverre bruker de fleste av dagens P2P-applikasjoner protokoller som er proprietære og inkompatible av natur. Hvert nettverk er et lukket samfunn, helt uavhengig av andre nettverk. Muligheten for å lage et stort nettverk som tilbyr et stort mangfold av tjenester med god tilgjengelighet og feiltoleranse forsvinner dermed. I stedet får vi flere mindre nettverk med delvis overlappende funksjonalitet. Litt av fordelen ved å samle terminaler i P2P-nettverk blir borte [4].

Frem til nå har iveren etter å utforske og teste ut mulighetene i P2P-teknologien helt overskygget viktigheten av interoperatibilitet og gjenbruk av programvare. Sun Microsystem så behovet av et felles P2P språk, og lagde derfor et prosjekt kalt JXTA. JXTA er egentlig et sett med protokoll-spesifikasjoner. Det er nettopp dette som gjør det så kraftfullt.

Dette kapittelet presenterer P2P-plattformen JXTA. I avsnitt 4.1 presenteres rammeverket i JXTA. Avsnitt 4.2 og 4.3 tar for seg konseptene og protokollene som utgjør JXTA. Til slutt ser vi på sikkerhet og Content Management System (CMS).

4.1 JXTA rammeverk

Navnet JXTA kommer fra ordet ”*juxtapose*”, som betyr å sidestille to enheter. JXTA definerer et sett med protokoller for P2P-nettverk. Protokollene er basert på XML (Extensible Markup Language) [53]. De beskriver komplekse operasjoner som oppdagelse av peer-er (like), endepunkt-ruting[16], oppsett av en forbindelse, enkel meldingsutveksling og utbredelse av nettverket gjennom *rendezvous* peer-er (mer om rendezvous i avsnitt 4.2). [22]

JXTA er et *Open Source* prosjekt, som drives av Sun Microsystems. Open Source vil si at kildekoden er åpen for alle, og at det er et *Community* (samfunn) som står for utviklingen av kildekoden på frivillig basis. Open Source er i følge Suns Simon Phipps:

“...best suited for developing foundational code. This is the theory behind Sun’s JXTA project, which serves as the Open Source foundation for P2P applications.”[17]

Prosjektet har som mål å lage en felles plattform for utvikling av P2P-applikasjoner. Konkret vil "felles plattform" si:

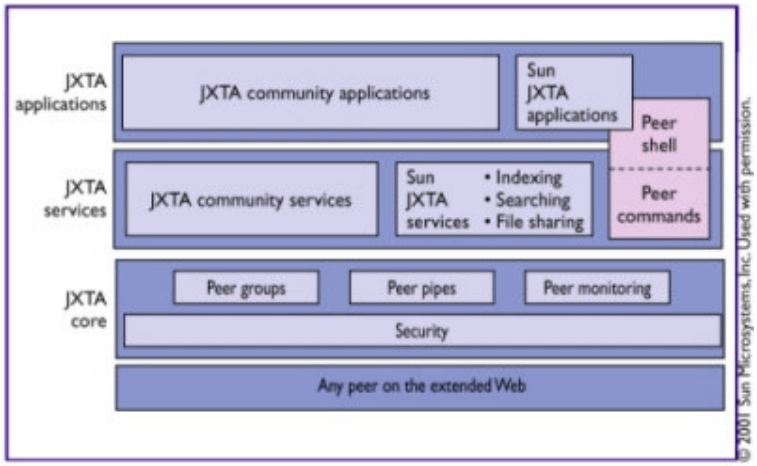
- **Interoperabilitet** – mellom forskjellige P2P-systemer og samfunn.
- **Plattformuavhengighet** – uavhengig av programmeringsspråk, operativsystem og nettverk
- **Alle steds nærværende**[17]– kunne brukes i alle typer terminaler med en prosessor og nettverksmuligheter.

[16] Et endepunkt er et nettverks-grensesnitt som brukes til å sende og motta data. Alle data som sendes over nettverket starter i ett endepunkt og ender opp i et annet.
[17] fritt oversatt fra det engelske ordet *ubiquity*

Det har hele tiden vært et ønske om å holde spesifikasjonen enkel og liten. **Enkel**, for å kunne gi rom til innovasjon blant programvareprodusenter [4], **liten**, for at JXTA skal kunne være alle steds nærværende. Mange områder som for eksempel sikkerhet (omtalt i avsnitt 4.4), ville ha krevd mye spesifisering, noe som igjen ville ført til at spesifikasjonen ble større. JXTA Community har derfor valgt å underspesifisere enkelte områder, og overlate problemstillingene til den enkelte utvikler. For at JXTA skal slå an er programvareprodusentene nødt til å ta i bruk JXTA. Hvis ikke disse ønsker å ta JXTA i bruk, vil sannsynligvis JXTA stort sett bare brukes av de som har bidratt til kildekoden og noen universiteter rundt omkring.

Figur 4-1 viser en typisk programvarearkitektur for en P2P applikasjon utviklet med JXTA.

- Det nederste laget, **kjernen**, tar seg av opprettelse av nye peer-er, ruting og administrasjon.
- Det midterste laget, **tjenester**, tar seg av tjenester som indeksering, søking og fildeling.
- Det øverste laget, **applikasjoner**, er applikasjoner som for eksempel e-post, auksjoner og lagringssystemer.



Figur 4-1 JXTA P2P programvare arkitektur (hentet fra [22] med godkjennelse fra artikkelforfatter). JXTA tilbyr et lag som utviklere kan lage tjenester og applikasjoner over.

4.2 JXTA konsepter

Som nevnt i innledningen til dette kapittelet er JXTA egentlig et sett med protokoll[18]-spesifikasjoner (mer om protokollene i avsnitt 4.3). For å underbygge disse protokollene har JXTA Community definert en del **konsepter** som peer, gruppe og melding. [22] og [36]

peer

En peer er et grunnleggende element i ethvert JXTA nettverk. En peer skal implementere kjerneprotokollene til JXTA og kunne operere uavhengig og asynkront med alle andre peer-er. En peer tilbyr applikasjons- eller nettverkstjenester, og skal være i stand til å annonsere dette på egenhånd. (Det finnes unntak fra dette, se kapittel 5 "JXTA for J2ME™").

En peer er mye mer enn en enkeltstående applikasjon som kjøres på en maskin som er tilkoblet Internett. En peer kan være en applikasjon som er distribuert over flere maskiner eller et program som kjøres på en PDA, som kobler seg til Internett via en mobiltelefon. En enkeltstående maskin kan til og med være ansvarlig for mange peer-er. En peer er derfor en veldig vid betegnelse. [4]

[18] Protokoll – Språket som brukes for å gjøre det mulig for datamaskiner å utveksle informasjon mellom hverandre

Peer-er deles gjerne inn i tre underkategorier: **enkle** peer-er, **rendezvous** peer-er og **ruter** peer-er. En peer kan opptre som en eller flere av disse. En **enkel** peer skal betjene en enkelt bruker på kanten av nettet. En enkel peer kan være atskilt fra resten av nettverket med en brannmur. Enkle peer-er er ikke ansvarlig for å håndtere trafikk på vegne av andre peer-er. En **rendezvous** er egentlig en samlingsplass eller møteplass. En rendezvous peer hjelper peer-er, som for eksempel en enkel peer, å oppdage andre peer-er og tjenester. En **ruter** peer tilbyr mekanismer for å kommunisere med peer-er som er atskilt fra nettverket av en brannmur eller av NAT (Network Address Translation).

peer-gruppe
En peer-gruppe er en samling med peer-er. JXTA definerer protokoller for å opprette, slutte seg til, og monitorere grupper. En gruppe tilbyr tjenester som oppdagelse av peer-er, medlemskap, tilgangskontroll, piper og spørringer.

Peer-grupper er en måte å lage en logisk oppdeling av fysiske nettverk på. Et eksempel på et fysisk nettverk kan være et universitet eller en stor bedrift. En logisk oppdeling av dette kan være i avdelinger eller kollokviegrupper. I JXTA er en peer gruppe en samling av peer-er som deler ressurser og tjenester. Peer-grupper er nøkkelen til å lage tjenester med høy tilgjengelighet og feiltoleranse.

piper (kommunikasjonskanaler)
Piper eller kommunikasjonskanaler representerer virtuelle forbindelser mellom peer-er. En punkt-til-punkt (*unicast*) pipe kobler sammen to peer-er. En *propagate* (utbredelse) pipe kobler en peer til flere lyttere (*listeners*). Peer-er som er koblet sammen ved hjelp av en pipe, er ikke nødvendigvis linket direkte sammen fysisk. Det kan godt tenkes at de er koblet sammen gjennom flere mellomliggende piper.

meldinger
Meldinger er data som utveksles gjennom piper mellom peer-er og endepunkter. JXTA definerer et konvolutt-format for alle meldinger. En peer kan definere sitt eget format på meldingene, så lenge det er i henhold til XML-spesifikasjonen. Skal to peer-er utveksle informasjon som gir mening, må begge forstå hverandres meldingsformat.

En melding består av en konvolutt og et innhold (*body*). Konvolutten består av et hode, avsenders adresse (Uniform Resource Identifier - URI), mottakers adresse (URI) og et valgfritt felt. Innholdet er av variabel lengde og inneholder meldingen som peer-en ønsker å sende.

kunngjøringer (*advertisements*)
Kunngjøringer er strukturerte metadata som beskriver ressurser i JXTA nettverket. Kunngjøringene beskriver egenskapene til en JXTA komponent, som en peer, peer-gruppe, pipe eller tjeneste. Alle peer-er og tjenester forstår kunngjøringer.

identifikator
JXTA bruker UUID (Unique Universal IDentifier), et 64 bytes felt, for å referere til en entitet (peer, gruppe, tjeneste). Identifikatorene er på URN format, (Uniform Resource Name), og blir puttet inn i alle kunngjøringer. UUID-en blir generert ved hjelp av en algoritme som sørger for stor sannsynlighet for unik identifikator både i rom og tid [36].

JXTA er et P2P rammeverk som går mye lenger en kun enkel meldingsutveksling. Det tar hånd om spørsmål rundt P2P fildeling, P2P applikasjonstjenester og samarbeidende distribuert prosessering. Prosjekter som det jobbes med i JXTA Community er *Voice over P2P*, tjeneste for å spille av musikk innenfor bestemte grupper, lokasjonstjenester, en spillplattform og mye mer. Mens JXTA protokollene er laget med tanke på å være plattform- og språkuavhengig, er referanse-implementasjonen til JXTA bygget på Java-plattformen. Per dags dato finnes det også JXTA implementasjoner tilgjengelig for J2ME (Java 2 Micro Edition), C som kjører på Linux, C som kjører på Windows, Perl, Python og Smalltalk. I tillegg er det laget versjoner av JXTA for SOAP[19], XML-RPC[20] og Java RMI[21].

4.3 JXTA protokoller

Grunnlaget i JXTA spesifikasjonen består av et sett med protokoller som er språkuavhengige, plattformuavhengige og ikke stiller krav til underliggende nettverk. Foreløpig er det definert seks protokoller. Protokollene gjør det mulig å oppdage ressurser i et nettverk, få status om andre peer-er, finne en passende rute til en annen peer og foreta forespørsler.

Under er en oversikt over de forskjellige JXTA protokollene:

- *Peer Discovery Protocol* – gjør det mulig å oppdage peer-er i nettverket.
- *Peer Resolver Protocol* – gjør det mulig for peer-er å sende og prosessere forespørsler.
- *Peer Information Protocol* – gjør det mulig for peer-er å få tak i informasjon fra andre peer-er i nettverket.
- *Rendezvous Protocol* – tar seg av spredning av meldinger mellom peer-er.
- *Pipe Binding Protocol* – gjør det mulig å knytte virtuelle kommunikasjonskanaler (piper) til et endepunkt (peer).
- *Endpoint Routing Protocol* – gjør det mulig å rute meldinger fra en peer (avsender) til en annen peer (mottaker).

4.4 Sikkerhet

Sikkerhetskravene til P2P-systemer er veldig lik kravene til andre datasystemer [23]. De tre dominerende kravene er konfidensialitet, integritet og tilgjengelighet. Disse tre kravene kan oversettes til funksjonelle krav som inkluderer: autentisering, tilgangskontroll, kryptering, sikker kommunikasjon og ”ikke-fornektelse”[22]. Den største forskjellen mellom et P2P-nettverk og et tradisjonelt klient/tjener-nettverk er at alle deltakere i nettverket er mottakere og tilbydere av informasjon. Et sikkerhetshull i et operativsystem eller i en applikasjon på èn enkel maskin kan gi inntrengere adgang til hele nettverket [3].

Sikkerhetskrav blir gjerne tilfredsstilt gjennom en sikkerhetsmodell eller sikkerhetsarkitektur som beskriver subjekter og objekter i systemet, og hvilke handlinger subjektene kan gjøre på objektene. Et eksempel på dette er operativsystemet UNIX. UNIX har en veldig enkel sikkerhetsmodell. Brukere er subjekter, filer er objekter. Om en bruker kan lese, skrive eller eksekvere en fil, kommer an på hvilke rettigheter brukeren har i følge rettighetsbeskrivelsen til filen.

JXTA definerer ikke noen høynivå sikkerhetsmodell. UUID-er har ingen ekstern betydning uten en navngivning- og bindingstjeneste. Hvis UUID-er blir bundet til eksterne navn, kan vi lage autentisering og digitale signaturer som forsikrer oss at vi snakker med rette vedkommende.

[19] SOAP – Simple Object Access Protocol
[20] XML-RPC – Remote Procedure Call, en lettvekts versjon av SOAP
[21] Java-RMI – Remote Method Invocation
[22] Med ikke-fornektelse menes at ingen brukere skal kunne benekte å ha sendt noe som de har sendt.

JXTA teknologien er nøytral til krypterings- og sikkerhetsalgoritmer. Den spesifiserer ingen sikkerhetsløsning. JXTA åpner for at forskjellige løsninger kan implementeres, og kan enkelt adoptere forskjellige sikkerhetsmodeller som bygger på velkjente og betrodde eksisterende teknologier. Nylig ble det mulig å benytte både TLS (Transport Layer Security) og X509.V3[23] digitale sertifikater på en slik måte at sentraliserte CA-er (Certificate Authority) ikke er nødvendig, men samtidig mulig å bruke.

Fordelen med fremgangsmåten JXTA Community har valgt er klar. Ved å ta utgangspunkt i eksisterende teknologier kreves ingen ny utvikling av programmer eller standarder. Velkjente teknologier og protokoller har dessuten mer tillit fra utenforstående. Protokoller som SSL og TLS er godt testet mot eventuelle sikkerhetshull, og både utviklere og hackere er klar over hvor vanskelig de er å "knekke". Åpenheten og fleksibiliteten i JXTA gjør at en hurtig kan adoptere nye teknologier inn i JXTA for å kunne tilby enda større sikkerhet og økt fleksibilitet. [64]

4.5 CMS (Content Management Service)

CMS (Content Management Service) tillater JXTA applikasjoner å dele innhold innenfor en peer-gruppe [52]. CMS administrerer det delte innholdet for en peer og lar applikasjoner få se igjennom og laste ned innhold fra andre peer-er. CMS realiserer P2P fildeling i JXTA.

Hvert eneste element som deles, blir representert med en unik identifikator og en kunngjøring (se Eksempel 4-1) som gir meta-informasjon om innholdet. Eksempel på meta-informasjon kan være navn, lengde, type og en beskrivelse av innholdet. CMS tilbyr også en protokoll basert på JXTA piper for å overføre innhold mellom peer-er.

```
<?xml version="1.0">
<!doctype jxta:contentAdvertisement>
<jxta:contentAdvertisement>
    <name>index.html</name>
    <cid>md5:1a8baf7ab82c8fee8fe2a2d9e7ecb7a83</cid>
    <type>text/html</type>
    <length>23983</length>
    <description>Web site index</description>
</jxta:contentAdvertisement>
```

Eksempel 4-1 CMS – kunngjøring [52]. En kunngjøring av en HTML-fil som heter `index.html`.

4.6 Oppsummering

I dette kapittelet har vi tatt for oss JXTA og sett litt på hvilke muligheter JXTA tilbyr. JXTA er en ny distribuert plattform designet for å løse en rekke problemer i moderne distribuert prosessering innen området som refereres til som *P2P computing*. Målet er å tilby interoperabilitet mellom maskiner i et nettverk, være plattformuavhengig og være alle steds nærværende (være tilgjengelig på alle terminaler med en prosessor).

[23] X509.V3 – standard for digitale sertifikater fra ITU-T. For mer informasjon se [42].

JXTA definerer en rekke konsepter som er vanlige i P2P-nettverk, og som derfor også er hovedkomponentene i JXTA plattformen. Konseptene er peer, peer-gruppe, piper, meldinger, kunngjøringer, endepunkt og sikkerhet. En peer er en node i et P2P nettverk, mens en peer-gruppe er en selvorganiserende samling av peer-er med felles interesser. Kommunikasjonen mellom to peer-er foregår i form av meldinger som sendes gjennom piper fra et endepunkt til et annet. Kunngjøringer er strukturer av metadata som beskriver nettverksressurser. Kunngjøringene blir publisert og utvekslet mellom peer-er for å oppdage og finne tilgjengelige ressurser. Enheter i nettverket blir identifisert ved en unik ID som er uavhengig av nettverk eller systemspesifikke notasjoner.

JXTA spesifikasjonen består av et sett med protokoller som er språkuavhengige, plattformuavhengige og setter ingen forutsetninger til underliggende nettverk. Protokollene gjør det mulig å oppdage ressurser i et nettverk, få status om andre peer-er, finne en passende rute til en annen peer og foreta forespørsler.

Neste kapittel tar for se JXME (JXTA for J2ME) et prosjekt som har som mål å tilby JXTA-funksjonalitet til trådløse terminaler med Java-støtte.

5 JXTA for J2ME™

Trådløse terminaler som mobiltelefoner og PDA-er blir mer og mer utbredt. I Norge finnes det omkring fire millioner mobiltelefoner (juni 2002) [46]. Mobiltelefonene blir mindre og mindre, men har samtidig fått mer og mer datakraft og minne. Batterikapasiteten er blitt bedre og skjermene har fått bedre oppløsning og flere farger. Behovet for applikasjoner til trådløse terminaler vil øke i tiden som kommer [1].

P2P applikasjoner har flere egenskaper som gjør at de egner seg godt i trådløse omgivelser. De kan tilby høy tilgjengelighet, stor feiltoleranse og dynamisk oppdage andre terminaler og innhold.

JXTA for J2ME, forkortet JXME, er JXTA developer Community sitt svar på dette. I april 2001 gikk startskuddet for prosjektet. JXME skal tilby JXTA-kompatible funksjonaliteter på begrensede terminaler ved å bruke CLDC og MIDP. JXME skal dekke alt fra de enkleste mobiltelefoner til mer avanserte PDA-er. JXME gjør det mulig for MIDP-terminaler å delta i P2P-aktiviteter med JXTA peer-er som kjører på PC-er.

Mulighetene og fleksibiliteten i JXTA har en pris, kompleksitet. En JXTA-peer har mange oppgaver som ruting og annonsering og må prosessere XML-meldinger på *socket*-nivå. En slik peer ville blitt altfor stor og kompleks til å kjøre på en mobiltelefon. Et annet problem er at det ikke er støtte for XML eller *sockets* i standard J2ME/MIDP implementasjoner. For å kunne gjøre JXTA tilgjengelig for mobiltelefoner, er en lettvekts utgave av de viktigste JXTA API-ene nødvendig. [25]

Utfordringer
En av de største utfordringene ved å utvikle applikasjoner for trådløse terminaler/mobiltelefoner, er de mange begrensningene som er nevnt i kapittel 3. Utviklingen skjer fort, men likevel er forskjellene fra datamaskiner store. Under er en oppsummering av begrensningene i dagens mobiltelefoner [59]:

- Mulighet for **persistent lagring.** På dagens mobiltelefoner kan det være så lite som 8kB, som skal deles av alle MIDlet-er som er installert på telefonen..
- **Runtime heap.** Normalt mellom 32kB – 64kB.
- **Båndbredde.** Båndbredden er veldig begrenset (se Tabell 3-3 side 36). Forsinkelsen i nettet er stor.
- **Prosessorkraft**. Normalt rundt 20MHz.
- **Batterikapasitet.** Batterikapasiteten er en begrenset ressurs.

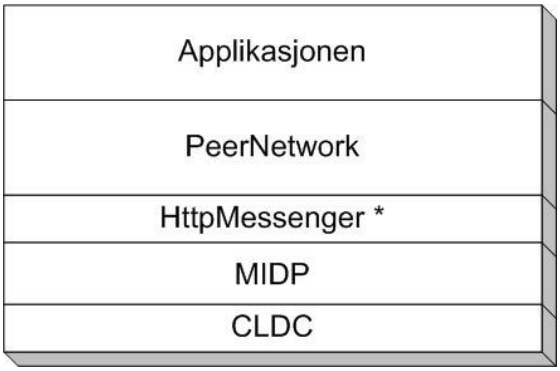
Disse begrensningene gjør at det ikke er mulig å kjøre alle typer applikasjoner på mobiltelefoner. Men mobiltelefonen har mange fordeler også. Fordi mobiltelefoner er små og mobile, egner de seg godt til å bære med seg rundt. Mobiltelefoner kan gi tilgang til informasjon fra det stedet brukeren befinner seg. Den kan være en slags portal inn mot nettverk hvor andre maskiner tar seg av prosessering og lagring. På grunn av sine begrensninger er telefonene avhengige av andre for å få jobben gjort. I JXTA løses dette ved at mobiltelefonene kobler seg til et *relé*, en PC i et stasjonært nettverk, som hjelper dem med oppgaver i JXTA-nettverket. Mer om JXTA relé i avsnitt 5.2.

Mål med JXTA for J2ME [59]
JXME skal:
- tilby P2P-infrastruktur for små begrensede terminaler
- tilby interoperabilitet med PC-er som bruker JXTA
- være enkel å bruke
- være liten nok til å kunne bli brukt på mobiltelefoner og PDA-er
- gjøre det mulig å lage applikasjoner som gir en god brukeropplevelse

JXME benytter versjon 1.0 av MIDP. MIDP 1.0 har dessverre en del begrensninger som er med på å prege JXME. Et eksempel på dette er de begrensede bibliotekene som følger med MIDP. Det finnes ingen XML-parser, bare støtte for utgående HTTP og ingen innebygget støtte for sikkerhet. MIDP 2.0, som ble presentert i 3.6, vil bli diskutert senere i kapittel 7 ”Diskusjon”.

5.1 Arkitektur

Som omtalt i avsnitt 3.1 har Java 2 Micro Edition to konfigurasjoner: CDC (Connected Device Configuration) og CLDC (Connected Limited Device Configuration). JXME har støtte for begge konfigurasjonene. Forskjellene mellom konfigurasjonene skjules (abstraheres) gjennom hjelpeklassen HttpMessenger. Applikasjonsutviklere får aldri direkte tilgang til HttpMessenger-klassens metoder. Utvikleren benytter metodene i PeerNetwork, som igjen bruker metodene i hjelpeklassen. I denne oppgaven ser vi på JXME med CLDC og MIDP i bunn. Figur 5-1 viser en oversikt over arkitekturen til en JXME applikasjon.



Figur 5-1 Arkitekturen til en JXME-applikasjon. (* HttpMessenger er en hjelpeklasse for PeerNetwork)

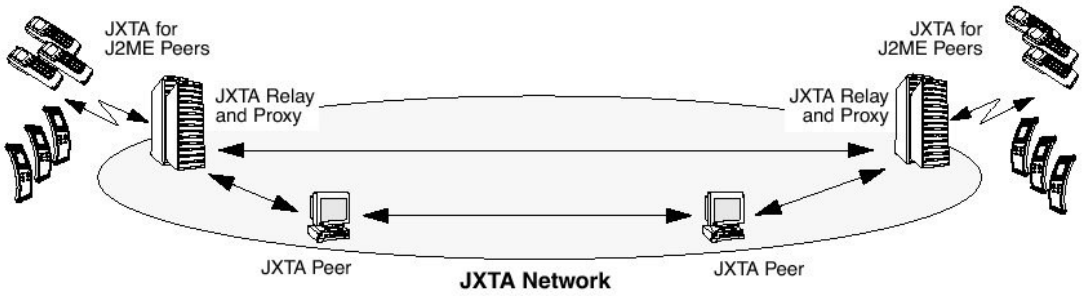
5.2 Relé (*Relay*)

På grunn av begrensningene i MIDP 1.0, kan JXME-peer-er bare opptre som **kant** peer-er (se Figur 5-2). Det vil si at de ikke kan tilby tjenester til andre medlemmer i en peer-gruppe direkte. JXME peer-er kan bare utføre enkle oppgaver. Tyngre operasjoner som søking og ruting må utføres av peer-er som kalles JXTA **relè**. En JXME peer blir et slags ”vindu” inn mot resten av JXTA-nettverket for sine brukere [15].

JXTA relé er en del av Java 2 Standard Edition (J2SE) versjonen av JXTA, og kjøres på en *rendezvous*. JXTA relèer opptrer som *proxyer*, det vil si at de handler på vegne av andre terminaler som for eksempel mobiltelefoner. De viktigste oppgavene relèer tar seg av er å:

- Tilby **oppdagelse** av bruker, gruppe og peer, slik at det kan opprettes piper, grupper og kommunikasjon.
- **Filtrere** bort unødvendige kunngjøringer, og trimme og optimalisere de kunngjøringene som sendes til JXME peer for å spare båndbredde.
- **Oversette** meldinger fra JXTA sitt XML-format til JXTA Binary Message format[24] og andre vei, for å tilby **interoperabilitet** mellom mobile peer-er og vanlige peer-er.
- **Rute** meldinger til og fra mobile peer-er.

[24] JXTA Binary Message format - benyttes i kommunikasjonen mellom mobiltelefon og JXTA relé



Figur 5-2 JXTA relè (Hentet fra [1])

De fleste JXTA peer-er kan opptre som relè. Ved å konfigurere *JXTA-shell*, *myJXTA* eller noen andre JXTA-applikasjoner, kan maskinene som kjører disse applikasjonene opptre som relèer.

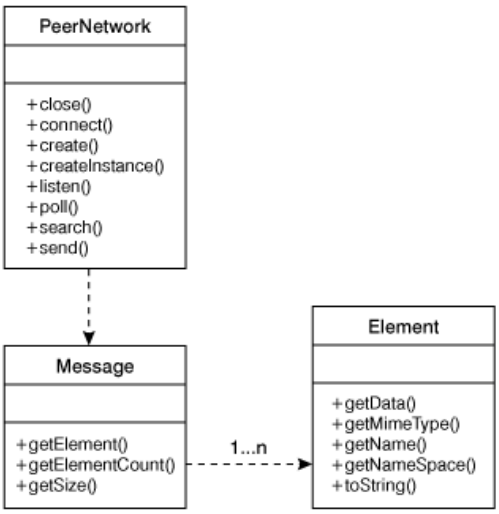
Alle meldinger som går i JXTA-nettverket er i XML-format. XML-meldinger er leselige for mennesker, noe som gjør det enklere for utviklere å programmere med. XML tilbyr god fleksibilitet, men har også en del ulemper. Blant annet er det nødvendig å ha en XML-parser for å kunne tolke meldingene. XML-meldinger har stor *overhead*, noe som fører til økt nettverkstrafikk og dårligere utnyttelse av båndbredden. Siden J2ME ikke har støtte for XML-parser og mobiltelefoner har begrenset og kostbar båndbredde tilgjengelig, egner XML seg dårlig for mobiltelefoner. JXTA-relèer trimmer derfor alle innkommende meldinger til et minimum.

Siden MIDP 1.0 bare har støtte for utgående HTTP-kall, er mobiltelefonen avhengig av å polle relèet for å undersøke om det er kommet noen nye meldinger. Pollingen skjer med et fast intervall som brukeren/utvikleren selv angir. Hvis det er en ny melding som venter, lastes denne ned i forbindelse med returen av det utgående HTTP-kallet. (mer om polling i kapittel 6)

5.3 Enkelt programmeringsgrensesnitt

API-et til JXTA for J2ME er enkelt og består kun av tre klasser (se Figur 5-3):

- **Message**: Denne klassen inneholder metoder for å lage og endre JXTA-Messages. En Message eller melding består av flere elementer. For mer informasjon se [59].
- **Element**: Denne klassen inneholder metoder for å konstruere og endre elementer i JXTA-Messages. Et element kan inneholde en pipeID, et navn, tekst eller data. Elementene lagres i en tabell som puttes inn i et Message objekt.
- **PeerNetwork**: Denne klassen inneholder operasjoner for å kunne kommunisere med JXTA-nettverket.



Figur 5-3 Klassene i net.jxta.j2me (hentet fra [25]). Hele JXME-API finnes i vedlegg IV.

Gjennom disse tre klassene tilbyr JXME JXTA-funksjonalitet som applikasjonsutviklere kan bruke. Eksempler på dette er kommunikasjon og oppdagelse av peer-er, piper og grupper. Under er en oversikt over de viktigste funksjonene:

- Oppdagelse av brukere (piper). Mulighet for å vedlikeholde og søke etter en begrenset liste med brukere.
- Oppdagelse av grupper (*search*). Mulighet til å finne en JXTA-gruppe og å bli med i denne.
- Oppdagelse av peer-er (*search*). Mulighet for å oppdage andre peer-er.
- Opprette piper (*create*). Lage punkt-til-punkt (*unicast*) eller *propagate* (utbredelse) piper.
- Opprette grupper (*create*). Mulighet for å opprette JXTA-grupper.
- Bli med i grupper (*listen*). Mulighet for å bli med i JXTA-grupper.
- Kommunisere (*send*, *poll*). Utveksle meldinger med andre peer-er.

Følgende funksjoner er det ikke støtte for i JXME per dags dato. Disse funksjonene vil det muligens komme støtte for i nyere versjoner av JXME, etter hvert som mobiltelefoner blir kraftigere:

- Relè-tjeneste. I fremtiden vil muligens en mobiltelefon kunne opptre som et relè ovenfor andre mer begrensede terminaler som for eksempel klokker.
- Ruting-tjeneste. Kunne rute meldinger og kunngjøringer for andre peer-er.
- *Rendezvous*-tjeneste. Kunne hjelpe peer-er med å oppdage andre peer-er og tjenester.
- Sikkerhet. Dette vil det muligens bli støtte for i forbindelse med integrasjonen av MIDP2.0 i JXME. MIDP 2.0 har støtte for HTTPS.

5.3.1 Oversikt over metodene

Metodene i Message og Element gir utviklere mulighet til å endre på elementene og meldingene som sendes og mottas. Metodene i PeerNetwork tar seg av all kommunikasjon mot JXTA-nettverket. Under er en oversikt over metodene i PeerNetwork og hva de forskjellige metodene gjør (hele JXME-API finnes i vedlegg IV):

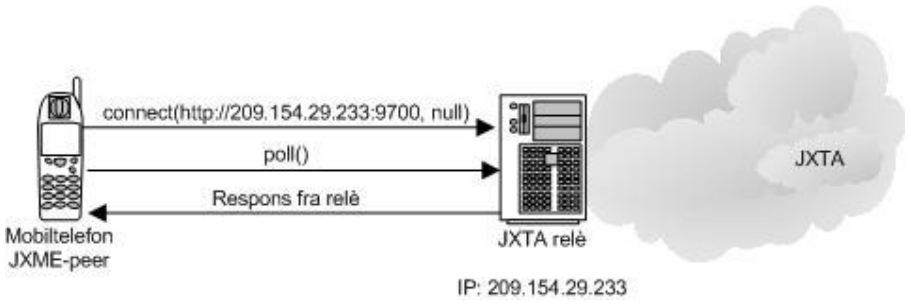
createInstance(PEER_NAME)
createInstance() oppretter en **peer** med et gitt navn, og returnerer en instans av PeerNetwork med et spesifisert peer navn. Navnet som peer-en får tildelt trenger ikke å være unikt.

```
peerNetwork = PeerNetwork.createInstance(PEER_NAME)
```

connect(String relayUrl, byte[] state)
connect() kobler en JXME peer til et relè med en spesifisert URL (IP-adresse og port). Det er mest vanlig å benytte port 9700 til JXTA-applikasjoner (se Figur 5-4).

```
persistentState = peerNetwork.connect(relayUrl, persistentState);
```

Det må alltid opprettes forbindelse med et relè før noen av de andre operasjonene kan kalles opp. `PersistenState`[25] kan brukes til å identifisere og motta meldinger som har kommet til en peer mens det har vært frakoblet relèet.



Figur 5-4 Kommunikasjon mellom JXME peer og JXTA relè. En JXME peer kobler seg først til et relè. Mobiltelefonen kommuniserer med relè over en HTTP forbindelse.

create(String type, String name, String arg)
create() brukes for å opprette en ny peer, gruppe eller pipe. Når create()-metoden kalles, sendes forespørselen videre til relèet. Responsen på metodekallet kommer tilbake til peer-en neste gang det foretas et poll(). Create() tar tre argumenter:

- type – angir hva som skal opprettes (peer, gruppe eller pipe)
- name – navnet på enheten som skal opprettes
- arg – et valgfritt felt avhengig av hva slags *type* som opprettes. For *pipe* kan utviklere angi hva slags type pipe (ende-til-ende eller *propagate*) de ønsker å opprette.

search(String type, String query)
search() lar deg søke etter peer-er, grupper eller piper i JXTA-nettverket. Type angir om det er peer, gruppe eller pipe du ønsker å søke etter. Query lar deg spesifisere hva du ønsker å søke etter. Query eller forespørselen er som regel et regulært uttrykk (*regex*), som for eksempel `TicTacToe*`, som gir deg alle enheter med navn som begynner med TicTacToe.

poll(int timeout)
poll() sender forespørsler til relèet om det finnes noen innkommende meldinger. Pollingen skjer periodisk. Bruker/utvikler bestemmer tidsintervallet mellom pollingene. Argumentet *timeout*, gjør det mulig å velge hvor lenge en peer skal vente på en respons. Verdien 0 betyr at peer-en skal vente for alltid. poll() returnerer en Message.

listen(String name, String id, String type) og close(String name, String id, String type)
listen() åpner en pipe for input. close() lukker en pipe for input. Metodene har tre argumenter:

- name – navnet på pipen du ønsker å lytte på eller lukke
- id – pipeID-en til pipen du ønsker å lytte på eller lukke
- type – angir om det er en ende-til-ende (unicast) eller propagate pipe

[25] Tilstanden til en forbindelse mot PeerNetwork representert ved en `byte[]`

send(String name, String id, String type, Message data)
send() brukes til å sende en melding ut på en gitt pipe med navn lik *name* og pipeID lik *id*. *Type* angir om det er en ende-til-ende (*unicast*) eller *propagate* pipe. *Data* er dataene som skal sendes ut på pipen. Dataene er av type **Message**.

5.3.2 Meldinger

Vi deler meldinger inn i utgående og innkommende meldinger (*Messages*). Utgående og innkommende meldinger er bygget opp på samme struktur, men har forskjellig type innhold. Som nevnt tidligere, består en melding av en tabell med elementer. Elementene kan være alt fra navn og pipeID til tekst og data.

```
Element(java.lang.String name, byte[] data, java.lang.String nameSpace,
java.lang.String mimeType)
```

Utgående meldinger
Utgående meldinger bygges opp slik utvikleren vil. Det er ingen krav til rekkefølgen eller innholdet i meldingene. Det eneste kravet er at alle peer-er som mottar meldingene kan forstå innholdet. Eksempel 5-1 viser hvordan en melding kan se ut.

Element	Navn	data	nameSpace	mimeType
1	MobsterName	(navnet på peer)	Null	null
2	MobsterCommand	(en kommando)	Null	null
3	ID	(ID til avsender peer)	Null	null

Eksempel 5-1 Eksempel på en melding med tre elementer

Innkommende meldinger
Innkommende meldinger kan enten være respons på en forespørsel gjort av JXME-peer, eller det kan være meldinger fra relè til JXME-peer. På en innkommende respons på forespørsel fra JXME-peer, vil oppbyggingen av meldingen være bestemt av utvikler (akkurat som Eksempel 5-1). Meldinger fra relè til JXME-peer, har derimot en fast oppbygning.

Alle metodekall av typen *send*(), *listen*(), *search*() og *create*() vil få returnert en "*requestID*", en unik identifikator. Ved hjelp av denne identifikatoren kan applikasjonen finne ut hvilket metodekall en respons fra relèet samsvarer med. Relèet legger til identifikatoren som et element i meldingen til JXME peer (se Eksempel 5-2). Et annet element som alltid følger med er "*response*". Dette feltet sier noe om hva slags type melding som kommer. Det finnes fem typer responsmeldinger:

1) Success
Hvis for eksempel en pipe er opprettet, vil relèet returnere en melding med response = *success* og med requestID lik den som ble returnert da create() ble kalt.

2) Error
Error returneres hvis det oppstod en feil under behandling av forespørsel fra peer. Et eksempel kan være hvis pipen som en melding skal sendes ut på ikke eksisterer.

Element	nameSpace	name	Data
1	Proxy	response	Success
2	Proxy	requestID	11
(Deretter kommer det påfølgende felt med ytterligere informasjon)			

Eksempel 5-2 Eksempel på respons fra relè

3) Info
Hvis en peer oppretter en pipe, vil peer-en først få en melding om dette gikk bra eller ikke (*success*/*error*). Hvis det gikk bra å opprette en pipe, vil informasjon om pipen returneres i en *info*-melding. I Eksempel 5-3 ser vi et eksempel på hvordan en slik info-melding kan se ut. Denne meldingen inneholder pipeID til pipen ved navn "testPipe" som akkurat er opprettet. Pipen er av type "*JxtaUnicast*" (ende-til-ende).

```
0 "proxy:response" "application/octet-stream" dlen=4 "info"
1 "proxy:requestId" "application/octet-stream" dlen=1 "0"
2 "proxy:type" "application/octet-stream" dlen=4 "PIPE"
3 "proxy:name" "application/octet-stream" dlen=12 "testPipe"
4 "proxy:id" "application/octet-stream" dlen=80
   "urn:jxta:uuid-
59616261646162614E50472050325033DE91EDF073014BB19BBDA4A5D344215B04"
5 "proxy:arg" "application/octet-stream" dlen=11 "JxtaUnicast"
```

Eksempel 5-3 Opprettelse av en pipe. Respons fra relè .

4) Result
Når en peer foretar et søk etter en pipe, peer eller gruppe vil resultatet være av type *result*. Meldingen vil inneholde navnet på enheten og identifikatoren til denne.

5) Message
Meldinger av type *message* er meldinger som er laget av applikasjoner tilknyttet nettverket. Hvis en mobiltelefon sender en melding med egendefinert innhold (tekst, bilde, ringetone) til en annen vil denne være av type *message*.

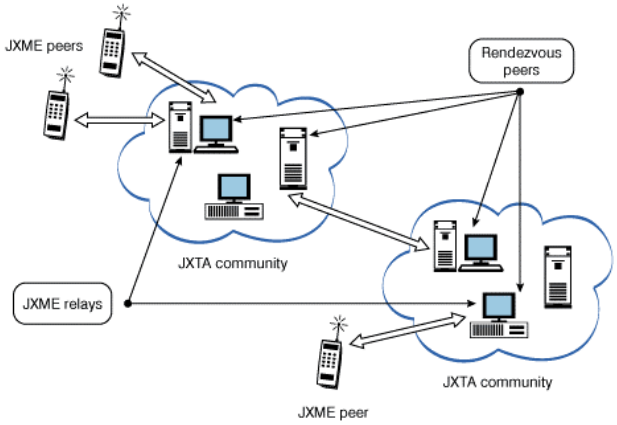
5.4 Bryter JXTA relèer P2P-prinsippet?
Filosofien i P2P-nettverk er at alle deltakere i nettverket skal opptre som likeverdige. Men hvor likeverdig er egentlig en mobiltelefon i forhold til en datamaskin? Hvor likeverdig er en peer som er avhengig av et relè for å kunne kommunisere med andre? Er relèet en like?

I følge JXTA Developer Community bryter ikke JXTA relèer med P2P-prinsippet. Argumentasjonen for dette er at to JXME peer-er som er koblet til hvert sitt JXTA relè, kan oppdage hverandre og kommunisere med hverandre Peer-to-Peer. Relèet tilbyr kun en tjeneste til mobiltelefonen, som den kan tilby tjenester for andre peer-er. I fremtiden vil JXME peer-er kunne velge relèer dynamisk. Slik det er nå, må brukeren av en applikasjon selv oppgi hvilket relè han ønsker å benytte. Konsekvensen av det er at mobiltelefonen er avhengig av èn sentralisert maskin for å kommunisere med nettverket.

Etter hvert som mobiltelefonene utvikler seg, vil muligens behovet for relè-tjenesten forsvinne, i hvertfall for mobiltelefoner sin del. (Det er mulig at andre begrensede terminaler fortsatt vil ha utbytte av denne tjenesten). Mobiltelefonene vil med tid og stunder muligens kunne kjøre fullversjonen av JXTA, og dermed stå for ruting og behandling av XML-meldinger på egenhånd. Hvorvidt dette kan være ønskelig vil vi diskutere i kapittel 8.

5.5 Oppsummering

I dette kapittelet har vi sett på JXTA for J2ME, forkortet JXME. JXME er et prosjekt som har som mål å tilby JXTA-funksjonalitet til trådløse terminaler med støtte for Java (se Figur 5-5). Som et resultat av begrensningene i MIDP, er implementasjonen avhengig av et *relè* som kan ta seg av det tyngste arbeidet for JXME-peer-en. Relèet tar seg av oversetting fra XML til JXTA Binary Message format, filtrerer bort unødvendige kunngjøringer, ruter meldinger og tar seg av oppdagelse av peer-er og grupper.



Figur 5-5 JXTA for J2ME (Hentet fra [25])

JXME tilbyr et enkelt API som bruker HTTP til å kommunisere med relèet. API-et består av tre klasser, en *Element*-klasse, en *Message*-klasse og en *PeerNetwork*-klasse.

Neste kapittel beskriver prototypen som er utviklet ved hjelp av JXTA for J2ME.

6 Prototypen

Hvordan vil P2P-fildeling fungere på en mobiltelefon? Hvilke muligheter finnes for å realisere dette ved hjelp av JXTA? For å kunne besvare disse spørsmålene har jeg i løpet av prosjektperioden utviklet en prototyp som realiserer P2P-fildeling på mobiltelefoner. Prototypen og tester som er gjort med denne gir grunnlag for drøftingen i kapittel 7 "Diskusjon".

Utviklingen av prototypen har vært med på å gi erfaringer med hva som er begrensningene i dagens teknologi, og hva som skal til for å overkomme disse begrensningene. Arbeidet med prototypen har også vært med på å gi ny kunnskap innen feltet P2P-fildeling for mobiltelefoner (se kapittel 7).

Prototypen som ble utviklet er et enkeltstående program, i form av en MIDlet. Den lar en bruker av mobiltelefon A søke etter, og laste ned filer direkte fra mobiltelefon B. MIDlet-en kan brukes til å teste ut og få erfaring med fildeling på mobiltelefoner. Flere av dagens mobiltelefoner har støtte for Java. Det er dermed teoretisk mulig å teste ut applikasjonen på en mobiltelefon. Prototypen kan gi oss en indikasjon på hvorvidt slike typer applikasjoner egner seg på mobiltelefoner. MIDlet-en er ikke et ferdig produkt, men et program som kan brukes til å demonstrere virkemåten og mulighetene i teknologien.

Ethvert system eller program bør ha et navn. Navnet bør enten lar seg forkorte eller si noe om systemet. Da prosjektet startet opp hadde ennå ikke bråket rundt *Napster* stilnet av. Dette prosjektet tar for seg mobiltelefoner og hvordan P2P fildeling kan realiseres på disse. Hvis mobiltelefon og Napster slås sammen, får vi *MOBster*. For enkelthets skyld kommer prototypen heretter til å bli omtalt som MOBster  .

6.1 Scenarier

P2P-fildeling på mobiltelefoner er et lite utforsket tema. Det har vært viktig å forhøre seg med kompetente personer innen de forskjellige fagområdene som prosjektet berører. I startfasen av prosjektet ble det i samarbeid med forskere fra Anarkistiske Nettsamfunn i Telenor FoU, foreslått en del mulige scenarier. Dette er personer som har kompetanse innenfor P2P, JXTA, Java 2 Micro Edition og mobiltelefoner.

Idèmyldringen resulterte i fire forskjellige scenarier som beskrives nærmere i avsnittene under. Hensikten med scenariene er å spenne ut et mulig løsningsrom for P2P fildeling på mobiltelefoner. Det skal også være med på å gi forståelse for problemområdet.

6.1.1 Mobiltelefonen som en fjernkontroll

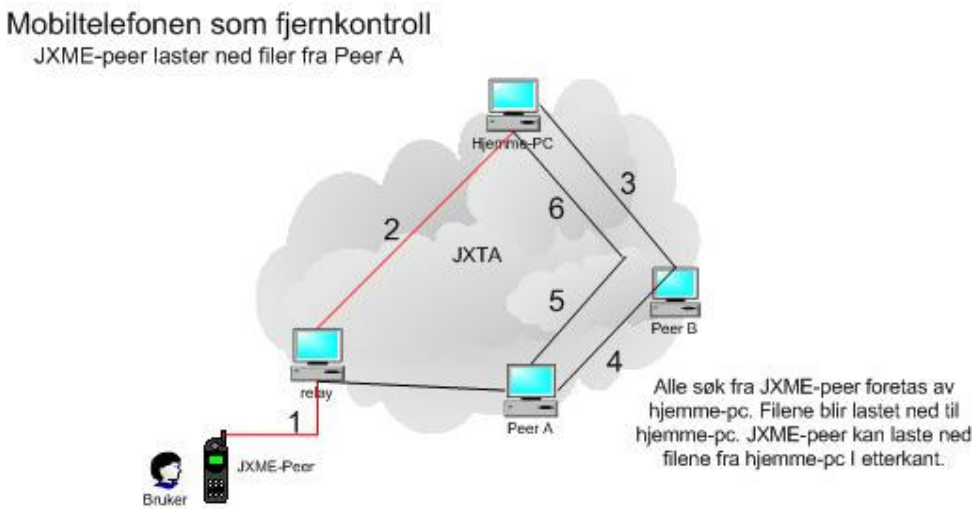
Da lagringskapasiteten på mobiltelefoner er begrenset, kan ikke brukere lagre alt de vil på mobiltelefonen. Nokia 6610 har som nevnt tidligere 725 kB tilgjengelig minne. En bruker må være svært selektiv med hensyn til hva slags filer han laster ned.

Kapittelet JXTA for J2ME™ redegjorde for hvordan JXTA-kompatible funksjoner kan tilbys på mobiltelefoner. På grunn av begrensningene som ligger i mobiltelefoner og J2ME, kan ikke mobiltelefoner delta i JXTA-nettverket på lik linje med datamaskiner. I kapittel 5 ble det nevnt at mobiltelefonen kan være et slags vindu inn mot JXTA-nettverket.

Mobiltelefonen har sine begrensninger, men den har også sine fordeler. Blant annet er den liten, og meget mobil. Den egner seg godt til å ligge i en veske eller jakkelomme, og følge brukeren over alt. Det første scenariet ser på muligheten av å gjøre mobiltelefonen til en slags utstrakt arm fra datamaskinen hjemme. Den vil da kunne fungere som en fjernkontroll som brukeren kan benytte for å administrere filer på sin egen maskin. Brukeren kan velge hvilke filer som skal deles, søke etter filer hos andre JXTA-peer-er og laste ned filer fra disse. Hvis brukeren ønsker å se på bilder eller lytte til musikk som ligger på hans hjemme-pc, kan filene overføres til mobiltelefonen.

Fordelen med denne løsningen er at det er hjemme-pc-en som står for all prosessering i forbindelse med fildelingen mot nettverket. Båndbredden til og fra pc-en er som regel bedre egnet til fildeling enn den båndbredden som tilbys over HSCSD eller GPRS. PC-en har stor lagringskapasitet og kraftig prosessor. Brukeren laster bare ned de filene som han ønsker å bruke til mobiltelefonen direkte fra sin egen hjemme-pc, og slipper samtidig å delta i et fildelingsnettverk, med all den prosessering det medfører.

I Norge er det i dag ikke ulovlig å laste ned musikk og filmer. Derimot er det ulovlig å gjøre musikk og filmer tilgjengelig slik at andre kan laste ned fra deg.



Figur 6-1 Scenario 1. Mobiltelefon som fjernkontroll.

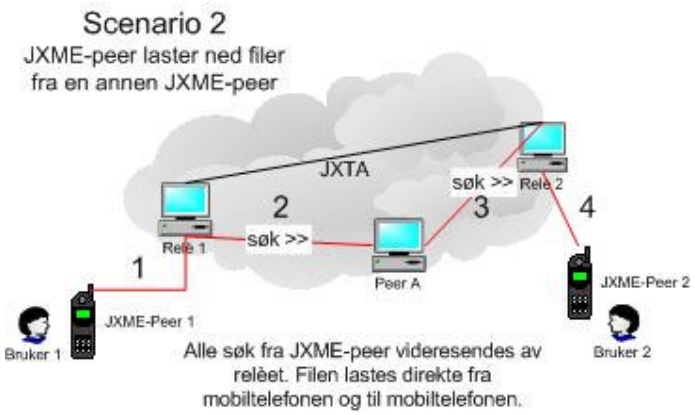
Figur 6-1 viser hvordan et tenkt scenario kan se ut. Mobiltelefonen sender en forespørsel til hjemme-pc, via et relè, der den ber pc-en om å laste ned en gitt fil. Hjemme-pc-en sender søket ut i JXTA-nettverket, og får treff fra Peer A. Filen lastes deretter ned til hjemme-pc-en. Brukeren kan velge om han vil laste filen ned til telefonen, eller bare la den bli liggende på hjemme-pc-en, avhengig av størrelse og nytte. Datatrafikk over mobilnettet er kostbart, og lagringsplassen på telefonen er begrenset.

6.1.2 Mobil-til-Mobil

Mobiliteten til mobiltelefoner åpner for en rekke nye muligheter og anvendelsesområder når det gjelder P2P fildeling. En typisk anvendelse av fildeling på mobiltelefoner kan være i en t-banevogn, i et klasserom eller når en venter på bussen. Mobiltelefonen er med overalt, og har stor dekningsgrad.

Det er søndag kveld, og du er på vei hjem fra byen etter å sett den siste James Bond filmen. T-banen skrangler av gårde forbi de samme stasjonene du alltid passerer. Du kjeder deg og har lyst til å høre på litt musikk. Låtene du allerede har lagret på mobiltelefonen er du lei av, så du bestemmer deg for å bruke MOBster til å søke etter den nyeste låten til A-ha. Du kobler deg opp mot relèet og noen tastetrykk senere er søket sendt av gårde. Søket blir mottatt hos mange MOBster-brukere rundt omkring. En stund senere begynner resultatene av søket å tikke inn. Det er visstnok mange som liker A-ha. Du velger en av filene og laster denne ned. Etter noen minutter er filen lastet ned og lagret på minnekortet som følger med telefonen. Du kan lene deg tilbake og lytte til A-ha mens enda en stasjon passeres.

I dette scenariet ser vi for oss at en mobiltelefon som er tilkoblet JXTA-nettverket via et JXTA-relè, skal kunne søke etter filer blant andre mobiltelefoner koblet til JXTA-nettverket.



Figur 6-2 Scenario 2. Mobil-til-mobil.

Figur 6-2 gir en oversikt over scenariet. Bruker 1 foretar et søk via sitt relè ut i JXTA-nettverket. Et søk sendes til relèet som oversetter søket til kunngjøringer. Disse sendes ut i JXTA-nettverket til alle som lytter på en forhåndsdefinerte pipeID. Når søkene/forespørslene mottas hos en JXME-peer, behandles søket lokalt før resultatet sendes tilbake til avsender. JXME-peer 2, som er tilkoblet relè 2 har den ønskede filen. Filen overføres via JXTA-nettverket til relè 1 og videre til JXME-peer 1.

6.1.3 JXTA-peer til JXME-peer

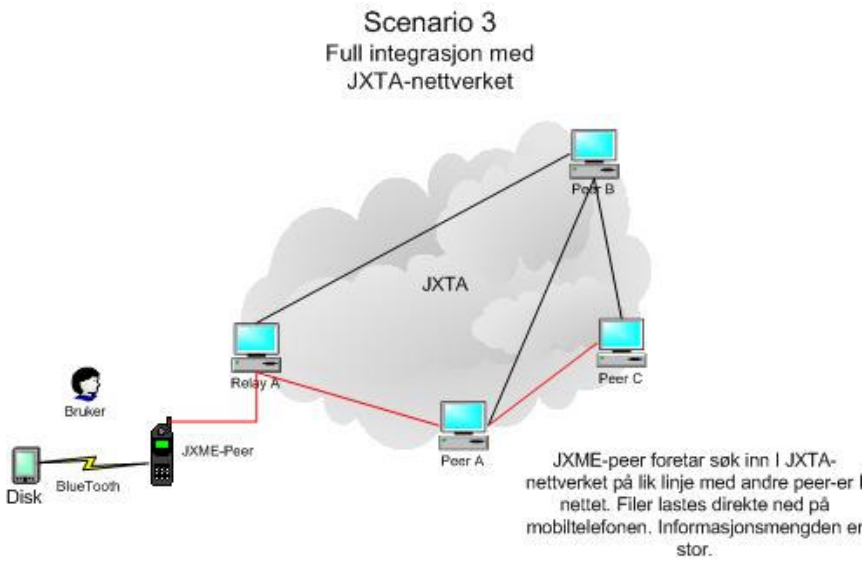
Informasjonsmengden på Internett er enorm. I følge [23] produseres det to exabyte eller omtrent $2x10^{18}$ byte hvert eneste år. Informasjonsmengden som er lagret i mobiltelefoner er per dags dato minimal. Det begrenser seg til litt musikk, bilder, ringetoner, SMS og telefonnummere. Med stadig flere mobiltelefoner utstyrt med kamera og med MMS (Multimedia Messaging Service) vil informasjonsmengden øke, men likevel være begrenset til enkelte filtyper.

Se for deg at du tar all informasjon som finnes i JXTA-nettverket og gjør den tilgjengelig for mobiltelefoner. Se for deg alle mulighetene som åpner seg! I dette scenariet ser vi for oss muligheten av å la mobiltelefonen (JXME-peer) få mulighet til å dele filer med alle peer-er i JXTA-nettverket. Mobiltelefonen får dermed tilgang til en stor mengde informasjon.

Med JXTA er dette mulig. JXTA er interoperatibel, den er plattformuavhengig og kan kjøres på stort sett alle enheter med en CPU (Central Processing Unit). JXME-peer-er vil kunne få tilgang til all informasjon som ligger i JXTA-nettverket. Dette vil kunne bety enormt for en tjeneste som MOBster.

Et problem med enkelte nye tjenester er at de krever et visst antall brukere eller en viss mengde informasjon for å kunne bli attraktive. En chat-tjeneste på Internett med kun to brukere er ikke særlig interessant. Et fildelingsprogram slik som KaZaA ville heller ikke vært veldig attraktivt å bruke om det kun hadde vært tre eller fire filer tilgjengelige for nedlastning. Slike systemer krever en viss mengde brukere eller en viss mengde informasjon for å bli attraktive å bruke. MOBster er ikke noe annerledes enn KaZaA. Uten brukere eller filer å dele, vil ingen benytte tjenesten. Hvis mobiltelefoner kunne fått tilgang til informasjonen som ligger lagret i JXTA-nettverket, kunne dette være med på å gi tjenesten en ”*flying start*”.

Vi tenker oss at filene som lastes ned til mobiltelefonen enten lagres på en minnebrikke eller overføres via BlueTooth til en bærbar disk brukeren har med seg i lomma. Sony har nylig kommet med en minnebrikke som kan lagre 1GB med data [65].



Figur 6-3 Scenario 3. Full integrasjon med JXTA-nettverket.

Figur 6-3 gir en oversikt over scenario 3. Mobiltelefonen deltar i nettverket på lik linje med JXTA-peer-er. Filer lastes ned til mobiltelefonen. Store filer lagres på ekstern disk eller på en minnebrikke.

6.1.4 Søk i JXTA, overfør med BlueTooth

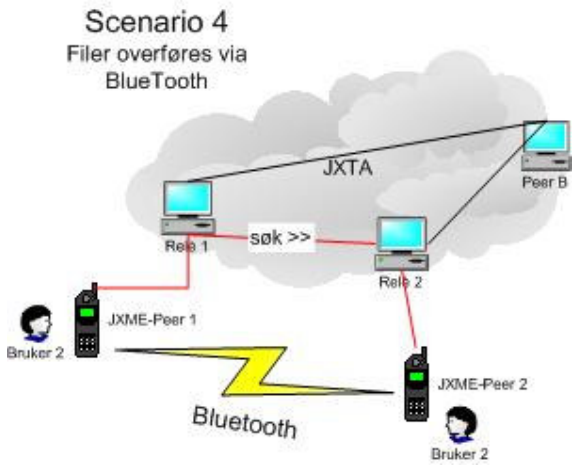
Båndbredden i GSM/GRPS er som nevnt tidligere begrenset og kostbar. Tabell 3-3 gir en oversikt over båndbredde og forsinkelse med de forskjellige teknologiene. Med HSCSD kan hastigheten komme opp mot 43 kbit/s. GSM/GPRS har en stor fordel, nemlig dekningsgraden. GSM-nettet er i dag godt utbygd og tilbyr god dekningsgrad over store deler av Norge.

Et økende antall av dagens mobiltelefoner er utstyrt med flere muligheter for kommunikasjon med omverden enn bare GSM/GPRS. I flere år har IR (InfraRed)-port vært standard på toppmodellene. IR gjør det mulig for to mobiltelefoner å kommunisere med infrarødt lys, så sant IR-portene kan "se" hverandre og ikke har for stor avstand. Flere av dagens telefoner er også utstyrt med BlueTooth [51]. BlueTooth opererer i 2,4 GHz-båndet. Den tilbyr opp mot 1 Mbit/s i overføringshastighet innenfor en radius på 10 meter. BlueTooth har den store fordelen at den ikke er avhengig av basestasjoner eller annen infrastruktur for å kunne kommunisere med andre. BlueTooth har heller ikke behov for synlig kontakt med terminalen den kommuniserer med. Personal Area Network (PAN) er navnet som brukes på slike type nettverk. De er spontane, brukes kun for anledningen og bare så lenge sesjonen varer.

GPS (Global Positioning System) [55] er et verdensdekkende posisjoneringssystem som gir deg koordinatene[26] til din posisjon. Ved hjelp av de 24 satellittene som svever rundt jorda, kan din posisjon bestemmes hvor som helst i verden, så lenge det er fri sikt mot himmelen. Med en GPS-mottaker innebygget i en mobiltelefon, vil det være mulig å fastslå hvor mobiltelefonen befinner seg til enhver tid.

[26] For ikke-militære GPS-mottakere er nøyaktigheten +/- 10 meter [55]

Tenk om en kunne kombinert det beste fra disse tre teknologiene? GSM tilbyr høy konnektivitet, BlueTooth tilbyr høy båndbredde og ingen overføringskostnader og GPS kan fastslå en mobiltelefon sin posisjon med rimelig stor nøyaktighet. I et mulig scenario foretar mobiltelefonen (JXME-peer) søk som vanlig over GSM/GPRS, via relè og inn i JXTA-nettverket. Søkene sendes utelukkende til andre mobiltelefoner. Alle mobiltelefoner som mottar forespørselen, sender en respons tilbake til avsender og hekter samtidig på sine lokasjonsdata. Når resultatene kommer i retur, sorteres resultatene etter nærhet i forhold til deg selv. Brukeren kan selv velge om han vil laste ned fra en som er i nærheten eller langt unna. Er den andre brukeren i nærheten kan brukeren enten oppsøke personen eller be om at filen skal overføres via BlueTooth (se Figur 6-4).



Figur 6-4 Scenario 4. Filer overføres via BlueTooth

På fester eller andre sosiale sammenkomster ser vi også for oss at BlueTooth kan brukes som bærer når det foretas søk. På den måten oppstår det et spontant nettverk blant deltakerne på festen. Disse kan da utveksle filer gratis ved hjelp av BlueTooth. Utveksling av filer *face-to-face* (ansikt-til-ansikt), gir en annen form for tillit enn når den det lastes filer fra er ukjent [14].

En annen teknologi som kan benyttes er WLAN. Terminaler som kombinerer WLAN og mobiltelefoni er under utvikling. Mange av dagens PDA-er har WLAN-kort. WLAN tilbyr større rekkevidde og båndbredde enn BlueTooth.

6.1.5 Oppsummering og drøfting

De fire scenariene over har vært med på å spenne ut et mulig løsningsrom for en P2P fildelingsapplikasjon på mobiltelefoner. Vi har sett på muligheter for å bruke mobiltelefonen til fjernkontroll av en hjemme-PC, dele filer mellom PC-er og mobiltelefoner og dele filer bare mellom mobiltelefoner. Selv om flere av disse scenariene ville være interessante å jobbe videre med, går enkelte av scenariene utenfor fokuset i prosjektet. Under er en kort oppsummering og drøftning av de enkelte scenariene.

Mobiltelefonen som en fjernkontroll
Det er mye som taler for en slik løsning. Derfor er det heller ikke noen stor overraskelse at det allerede finnes et slikt system (se ”Relatert arbeide - Apeera” avsnitt 7.6). Fordi mobiltelefonen er så begrenset er det en god idé å la en annen maskin gjøre jobben på vegne av mobilen. Mobiltelefonen tilbyr mobilitet og tilgjengelighet, datamaskinen tilbyr lagringskapasitet og prosessorkraft. Den type fildeling som beskrives i dette scenariet vil kunne passe inn i en vid definisjon av P2P-fildeling for mobiltelefoner. Mobiltelefonen er indirekte med i fildelingen, fordi den styrer hvilke filer som skal deles og kan søke etter filer. I scenariet deltar mobiltelefonen kun som en fjernkontroll og ikke som en *peer* eller like i fildelingen. Scenariet faller derfor utenfor vår definisjon av fildeling for mobiltelefoner.

Mobil-til-Mobil
Scenario to beskriver situasjoner hvor P2P-fildeling på mobiltelefoner kunne vært interessant, og litt om hvordan dette kan løses. Mobiltelefonene får kontakt med hverandre via relèer tilkoblet JXTA-nettverket. Filene som brukerne deler ligger lagret på mobiltelefonen.

Scenariet tar for seg fildeling mellom mobiltelefoner der mobiltelefonen står for lagring av filer og behandling av søk. Dette passer godt med vår definisjon av P2P fildeling for mobiltelefoner. Det er scenario to som kommer til å ha fokus i resten av oppgaven.

JXTA-peer til JXME-peer
Fordelene er mange ved å gjøre det mulig for mobiltelefoner (JXME-peer-er) og dele filer med stasjonære JXTA-peer-er. Som vi var inne på tidligere, vil den store informasjonsmengden som allerede finnes i JXTA-nettverket, være til stor nytte og hjelp for en mobil fildelingsapplikasjon.

Problemet per i dag med denne løsningen er at fildeling i JXTA skjer gjennom CMS (Content Management System, se avsnitt 4.5). Det er ikke støtte for CMS i JXME. Det finnes heller ingen metoder for å kommunisere med CMS på JXTA-peer-er fra JXME. Det er derfor ikke mulig å søke etter og laste ned filer fra JXTA-peer-er med dagens versjoner av JXTA og JXME. JXTA Community har startet et prosjekt som skal jobbe med å tilby CMS-funksjonalitet gjennom et ”CMS-lag” for JXME-peer-er.

For å realisere dette scenariet ville vi vært nødt til lage funksjonalitet som gjør JXME-peer-er i stand til å kommunisere med CMS på JXTA-peer-er. En måte å løse dette problemet på kunne vært gjennom en *servlet* som kjører på relèet. Alle spørringer mot JXTA-peer-er, kunne blitt sendt til servlet-en, som kunne oversatt disse til et format som JXTA-peer-er forstår. Responsene fra JXTA-peer-er ville gått via servlet og relè tilbake til mobiltelefonen. Servlet-en og relèet ville da til sammen utgjøre en utvidet relè-tjeneste. En slik løsning ville krevd mye programmering, og ville ha vært litt utenfor fokusområdet i oppgaven. Scenariet er interessant, og vil derfor være et forslag til videre arbeid.

Det er mulig å kjøre JXME-applikasjoner som *applet*-er på en webside. Å realisere programmet som en applet i tillegg til en MIDlet kan være med på å øke nytten av programmet. Applet-en vil bli en slags mellomting mellom programmet som kjører på en mobil og en JXTA-peer som deler filer med CMS. Applet-en vil oppføre seg som en meget ressurssterk mobiltelefon. Muligheten for å kjøre MOBster som en applet vil være et interessant forslag til videre arbeid.

Søk i JXTA, overfør med BlueTooth
Mange av dagens mobiltelefoner har BlueTooth. BlueTooth åpner for en rekke spennende muligheter. Dagens versjon av JXME egner seg ikke til å kjøre over BlueTooth. JXME-peer-er er avhengig av et JXTA relè for å kommunisere med hverandre og resten av JXTA-nettverket. Skulle en mobiltelefon koblet seg til et relè via BlueTooth vil det i praksis si at mobiltelefonen måtte vært innen 10 meters radius fra en datamaskin med BlueTooth-kort som kjører relè-tjenesten. Dette ville ha innskrenket mobiliteten til brukeren.

Et annet problem med dette scenariet er at GPS er lite utbredt i dagens mobiltelefoner. Uten GPS vil det være vanskelig å fastslå posisjonene til mobiltelefonene.

Scenariet er spennende. Fildeling mellom mobiltelefoner i et slikt spontant nett som BlueTooth lager, kan være veldig interessant i blant annet klasseroms-situasjoner, i sosiale sammenkomster og på arbeidsplasser. Dessverre er scenariet vanskelig å implementere med dagens terminaler og JXTA. Scenariet er interessant, og vil derfor være et forslag til videre arbeid.

Av hensyn til tidsforbruk har det i løpet av prosjektperioden blitt tatt valg som har avgrenset prosjektet. Scenario to ligger tett opptil det vi definerer som P2P fildeling for mobiltelefoner. Vi tror at en prototyp basert på dette scenariet vil kunne lære oss mest om P2P fildeling for mobiltelefoner og de aktuelle teknologiene som finnes i dag. I neste avsnitt vil vi gi en oversikt over systemet som ble laget. En uformell, og en formell kravspesifikasjon som beskriver systemet ved hjelp av UML (Unified Modeling Language) *use cases* ligger vedlagt (se vedlegg I).

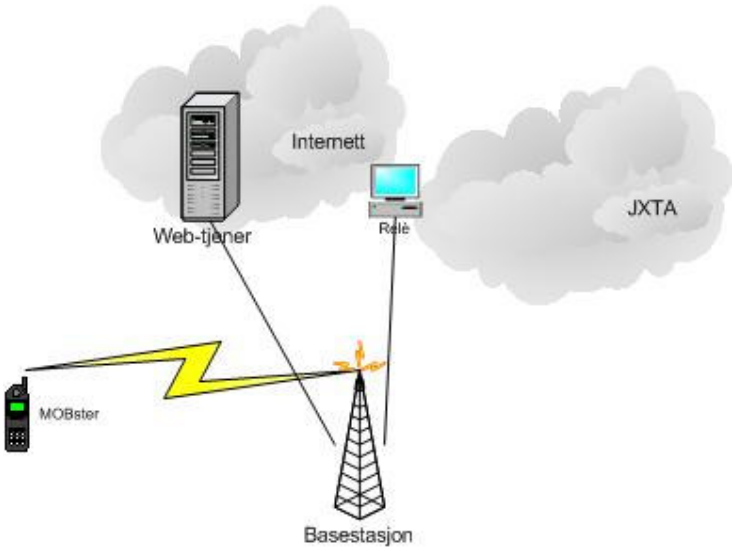
6.2 Systemoversikt

Som nevnt i innledningen, gjøres dette prosjektet i samarbeid med forskningsprogrammet Anarkistiske Nettsamfunn i Telenor FoU. Prosjektet er en del av forskningsprogrammet sitt arbeid med JXTA. Ved å benytte JXTA som plattform for programmet, var vi avhengig av et JXTA relè for å få applikasjonen til å fungere. Det var også nødvendig å innføre en webtjener for å få en tilfredsstillende funksjonalitet (se under avsnittet ”Webtjener”). MOBster består derfor av tre hovedelementer: en mobiltelefon (JXME-peer), et relè og en webtjener (se Figur 6-5).

Mobiltelefonen
Programmet skal kunne kjøres på en mobiltelefon. Mobiltelefonen må ha støtte for J2ME, både CLDC og MIDP 1.0. Det må være nok minne på telefonen til å kunne installere JXME og applikasjonen. Dagens telefoner har mer enn nok minne til en slik oppgave. Mobiltelefonen må kunne koble seg opp mot Internett, enten via GSM, GPRS eller andre teknologier. Applikasjonen kjøres som et enkeltstående program i en MIDlet suite på mobiltelefonen.

JXTA Relè
Alle JXME-peer-er er avhengig av et JXTA-relè for å kunne kommunisere med JXTA-nettverket. Relèet tar seg av oversetting av XML-meldinger, ruting av meldinger, oppdagelse av andre peer-er og filtrering av innkommende meldinger. Relèet må være plassert på en maskin som har en IP-adresse som kan nås fra Internett. Relè-tjenesten kan leveres av *JXTA shell*, *myJXTA* eller andre JXTA-applikasjoner [59]. Mer om relèet finnes i avsnitt 5.2.

Webtjener
Mobiltelefoner har begrenset lagringsplass og begrensede muligheter til å behandle filer. I kapittel 3 så vi på begrensningene i MIDP. Det er ikke mulig for en J2ME applikasjon å få tak i filer eller annen informasjon utenfor sin egen MIDlet suite. Begrensningene er laget av hensyn til sikkerhet. Ingen skal ha mulighet til å lage ”ondsinnede” programmer som henter ut personlig informasjon fra din mobiltelefon.



Figur 6-5 Systemoversikt

I tillegg har ikke alle mobiltelefoner muligheten til å få overført filer via en seriell kabel. Hvis ingen mobiltelefoner i et nettverk har noen filer å dele, vil fildeling fungere dårlig. Det vil derfor være nyttig å ha tilgang til en webtjener som brukeren kan laste ned filer fra. Dette kan sees på som en måte å initialisere programmet på. Første gang programmet starter opp, lastes noen forhåndsbestemte filer ned til telefonen. Deretter brukes ikke denne funksjonen mer. På sikt kan en slik funksjon som webtjeneren har, byttes ut med en hjemme-pc eller en JXTA-peer.

6.3 Design

JXTA og spesielt JXME er foreløpig en forholdsvis umoden teknologi. Det er ennå ikke lansert en endelig versjon som er klar for markedet. Jevnlig slippes det stabile utgaver av plattformen, men også disse har det vært en god del problemer med. Gjennom hele prosjektperioden har det vært en rekke endringer av implementasjoner både i JXME og i JXTA-relè. En virkning av dette er at det gjennom prosjektperioden ble funnet direkte feil i selve JXTA for J2ME plattformen (mer om dette i kapittel 7 ”Diskusjon”).

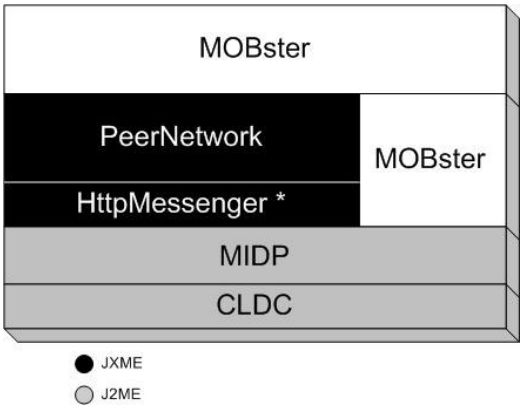
Det foreligger lite informasjon om utvikling av JXME-applikasjoner. I startfasen var det eneste tilgjengelige materialet API-et og et komplisert chat-eksempel. Både i API-et og i chat-eksempelet var det feil og mangler. Dette har ført til at det har tatt tid å komme inn i teknologien og få den nødvendige forståelsen for å kunne utvikle en applikasjon.

Det var derfor vanskelig å lage et godt design av systemet i forkant av implementeringen. Et overordnet bilde av det som skulle lages i tillegg til scenariet har dannet grunnlaget for designet. Utviklingen har båret preg av mye prøving og feiling, spesielt fordi enkelte av feilene har blitt generert av selve plattformen.

I neste avsnitt skal vi se nærmere på programvarearkitekturen som er benyttet i MOBster. Deretter skal vi se nærmere på klassediagram og metoder. I avsnitt 6.3.3 beskriver vi systemet gjennom sekvensdiagram. Delkapittelet avsluttes med avsnitt 6.3.4, som sier noe om oppbyggingen av brukergrensesnittet.

6.3.1 Arkitektur

MOBster har en to-delt arkitektur vertikalt, se Figur 6-6. Felles for begge deler er J2ME med MIDP (Mobile Information Device Profile) og CLDC (Connected Limited Device Configuration). MOBster kommuniserer enten direkte med MIDP og CLDC eller via `PeerNetwork` og `HttpMessenger`. All P2P-aktivitet og all kommunikasjon mot JXTA går via JXME (svart) og metodene i `PeerNetwork`. I oppstartsfasen derimot, bruker MOBster HTTP-kall direkte via J2ME (grå) mot en webtjener.



Figur 6-6 MOBster arkitektur. (* HttpMessenger er en hjelpeklasse for PeerNetwork).

6.3.2 Klassediagram

I avsnittet "Ytelsesdrevet programmering" ble det tatt for seg en del tips til hvordan utviklere kan gjøre en J2ME-applikasjon mest mulig effektiv. En mobiltelefon har en liten CPU og har begrenset batterikapasitet. Utviklere må forsøke å gjøre programmet effektivt og raskt, samt begrense bruk av prosessor og da også strøm.

Tipsene fra avsnitt 3.7.3 har ført til at alle metodene i MOBster ligger samlet i en stor klasse med en liten hjelpeklasse inni. Metodene er store, for å begrense antall metodekall. I Tabell 6-1 er de viktigste metodene i mobster.java listet opp (de resterende finnes i Javadoc-en[27] som er vedlagt oppgaven, se vedlegg III).

Metode	Beskrivelse
`poll()`	Poller relèet med et fast intervall. Ser etter innkommende meldinger. Gjør en grovsortering av innkommende meldinger.
`sendSearch(String searchString)`	Sender søk (`searchString`) ut på en kjent pipe som alle MOBster-applikasjoner lytter på.
`sendData(String filename, String reciever, String senderPipe)`	Sender en gitt fil `filename` til en mottaker `reciever` som lytter på en pipe med adresse `senderPipe`.
`sendResult (String msg, String reciever, String senderPipe)`	Sender et svar `msg` tilbake til en mottaker `reciever` som lytter på en pipe med adresse `senderPipe`.
`parseCommand(String message, String senderName, String senderPipe)`	Analyserer kommando-feltet i innkommende meldinger, og tar nødvendig aksjon. (Eksempel: Presenterer et søkeresultat på skjerm)
`loadImages(String url)`	Brukes i oppstarten av programmet. Den kobler seg opp mot en webtjener med adresse `url` og laster ned filer til mobiltelefonen.

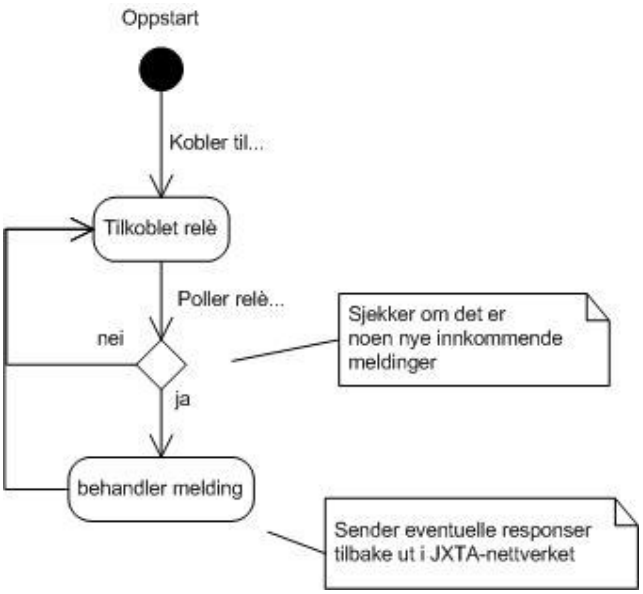
Tabell 6-1 Oversikt over de viktigste metodene i klassen mobster.java

[27] Javadoc er et verktøy som følger med Java Development Kit (JDK) for å generere API dokumentasjon.

6.3.3 Sekvensdiagram

MOBster sin oppførsel kan deles inn i to faser. Første fase er tilkoblingen til relèet. Dette er det første som gjøres før applikasjonen kan brukes. Fase nummer to, som starter etter tilkobling til relèet og varer helt til applikasjonen stoppes, er pollingen. Under tilkoblingen til relèet forsøker MOBster å nå relèet på en bestemt IP-adresse og en bestemt port (som brukeren selv kan angi under konfigurasjonen). Etter at forbindelsen er satt opp, opprettes en innkommende pipe til peer-en ved at create()-metoden kalles. JXME-peer-en får tildelt en egen URI til pipen sin. Etter dette går applikasjonen over i polle-modus (se Figur 6-7).

I polle-modus går programmet i en evig løkke. Etter et gitt tidsintervall poller relèet for å sjekke om det er noen innkommende meldinger som venter. Poll() gjør deretter en grovsortering av de innkommende meldingene, og sørger for at informasjonen havner der den skal i programmet.



Figur 6-7 Tilstandsdiagram for MOBster

Figur 6-9 og Figur 6-10 viser UML-sekvensdiagram for de to fasene tilkobling og polle-modus.

Tilkobling

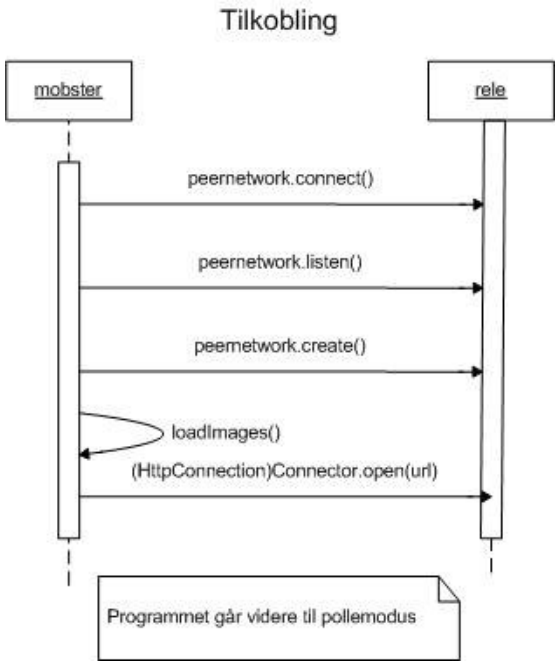
Mobiltelefonen må være tilkoblet relèet for å kommunisere med JXTA-nettverket. Tilkoblingen vil normalt være det første som skjer før brukeren tar programmet i bruk. Hvis brukeren kun ønsker å se på sine egne filer, er det ikke nødvendig å koble seg opp mot et relè. Filene ligger lagret lokalt på mobiltelefonen. Etter at brukeren har valgt å koble seg til, vises det en teller på skjermen. Telleren angir hvor lang tid det tar å koble seg til relèet.

Først kaller JXME-peer-en opp *connect()*-metoden. Connect-metoden forsøker å koble mobiltelefonen opp mot et relè på en gitt IP-adresse og port. Hvis tilkoblingen av en eller annen grunn ikke lykkes, vises det en feilmelding på skjermen. Etter at JXME-peer er koblet til relèet, kaller mobiltelefonen *listen()* på pipeID-en til MOBster (se Figur 6-8). Når listen() er kalt på en pipeID, vil peer-en som har utført kallet motta alle meldinger som sendes ut på den bestemte pipen. Relèet sørger for at alle meldinger kommer dit de skal. Pipen er en *propagate* (utbredelse) pipe, som egner seg til å sende forespørsler som skal til alle MOBster-peer-er.

```
urn:jxta:uuid-59616261646162614E50472050325033106963536861726 52D5B46696C65436104
```

Figur 6-8 PipeID til MOBster

MOBster sin pipeID er hardkodet i applikasjonen, og er derfor kjent av alle MOBster-peer-er. Med denne pipen kan peer-ene spre søk til alle MOBster-peer-er. For å kunne sende meldinger direkte (en-til-en) mellom to peer-er, må alle MOBster-peer-er i tillegg opprette en egen pipe, som er unik. Pipen opprettes ved å kalle *create*()-metoden. I motsetning til MOBster-pipen er denne en *unicast* (endepunkt-til-endepunkt) pipe, som egner seg godt til å sende data mellom to peer-er. PipeID-en til den nye pipen, opprettes av relèet, som returnerer ID-en tilbake til JXME-peer ved neste poll.



Figur 6-9 UML-sekvensdiagram for tilkoblingen. Først kobler mobiltelefonen seg til relèet med connect(). Deretter kalles listen() på pipeID-en til MOBster. Etter dette oppretter mobiltelefonen en innkommende pipe, som skal brukes til å motta meldinger. Til slutt initialiseres telefonen, ved at loadImages()[28] laster ned forhåndsbestemte bilder til telefonen.

Figur 6-9 viser tilkoblingen gjennom et UML-sekvensdiagram. Alle metoder som hentes fra JXME-API, starter med *peernetwork*. Metoden Connector.open er en CLDC-metode.

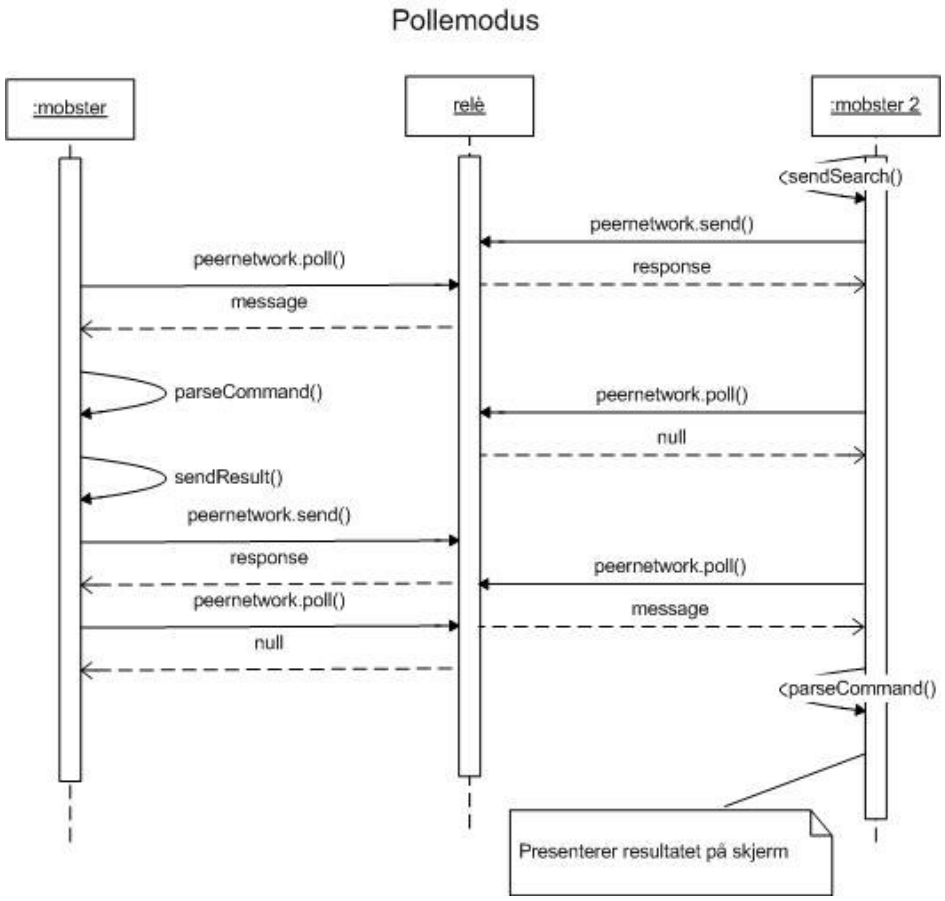
Etter at mobiltelefonen er koblet til relèet og pipen er opprettet, vil mobiltelefonen laste ned noen forhåndsdefinerte filer fra en webtjener. Dette gjøres bare første gang applikasjonen brukes.

Pollemodus

Etter tilkobling og initialisering går programmet over i pollemodus. Applikasjonen vil fortsette å polle helt til programmet avsluttes. Siden MIDP 1.0 bare har støtte for utgående HTTP-kall, er mobiltelefonen avhengig av å polle relèet for å laste ned innkommende meldinger. Pollingen skjer med et fast intervall som brukeren selv angir.

[28] *loadImages*() er en metode som kalles lokalt på mobiltelefonen, og som ikke benytter seg av nettverksforbindelsen

Figur 6-10 viser et UML-sekvensdiagram for pollemodus. I diagrammet foretar mobster 2 et søk. Søket sendes ut på MOBster-pipen ved hjelp av *send*(). Søket spres ut i JXTA-nettverket. En annen mobiltelefon (mobster) er tilkoblet et relè (ikke nødvendigvis det samme relèet som mobster 2). Mobiltelefonen poller relèet etter innkommende meldinger. Denne gangen venter det en melding. Meldingen lastes ned, og analyseres ved hjelp av *parseCommand*()[29]. Meldingen inneholder et søk fra en annen peer. Søket behandles, og resultatet sendes tilbake til mobster 2 via relèet.



Figur 6-10 UML-sekvensdiagram for pollemodus. Mobster 1 og mobster 2 er begge to tilknyttet et relè (ikke nødvendigvis det samme relèet).

Mobster 2 poller også relèet jevnlig etter innkommende meldinger. Noen poll senere kommer det en melding. Meldingen analyseres som vanlig med *parseCommand*(). Denne gangen er meldingen en respons på søket som ble sendt tidligere. Resultatet blir presentert for brukeren på skjermen.

6.3.4 Brukergrensesnitt

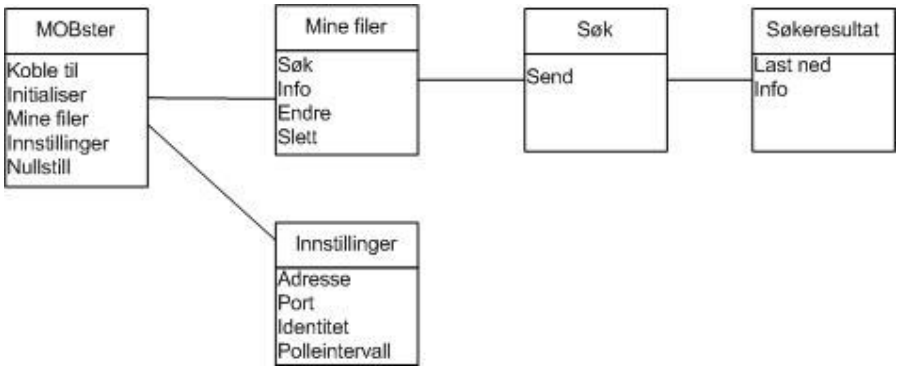
På begrensede terminaler med små skjermer og få input-muligheter er utformingen av brukergrensesnittet viktig. Et fornuftig brukergrensesnitt bygget opp på logiske sekvenser har mye å si [18]. I stedet for å presentere all informasjon i ett skjermbilde, deles informasjonen opp, og brukeren veiledes gjennom det som skal gjøres ved hjelp av flere påfølgende skjermbilder. Dette fører til mindre scrolling, og mindre skriving.

[29] *ParseCommand*() er en metode som ble utviklet i prosjektet

I denne utgaven av prototypen har ikke brukergrensesnittet vært særlig høyt prioritert. Fokuset har ligget på funksjonalitet fremfor brukervennlighet. I en eventuell ny versjon av MOBster ville brukergrensesnitt og brukeropplevelse hatt et sterkere fokus.

Mål for presentasjon i små display:
A) Minst mulig scrolling.
B) Minst mulig skriving.
C) Innholdet må være i fokus. Minst mulig effekter og animasjoner. [18]
D) Lage enkle logiske sekvenser av kompliserte funksjoner [18]. Dele opp funksjoner over flere påfølgende skjermbilder.

Figur 6-11 viser en oversikt over brukergrensesnittet til MOBster. Påfølgende skjermbilder er koblet sammen med en linje. Hvis en bruker for eksempel sender et søk, vil skjermbilde "Søkeresultat" vises på mobiltelefonen. I alle skjermbilder har brukeren mulighet til å trykke på en "tilbake"-knapp, som tar brukeren et skjermbilde tilbake (à la navigering på nettsider). Det er viktig for en bruker å kunne gå tilbake hvis han er usikker på det han gjør eller han angrer på de valgene han har gjort.



Figur 6-11 Oversikt over brukergrensesnittet i MOBster. Overskriften som er over streken er navnet på skjermbildet. Elementene under streken er menyelementer.

Skjermbilder

Utviklingen av MOBster foregikk på J2WTK (Java 2 Micro Edition Wireless Toolkit). Ved hjelp av emulatorene som finnes i programmet kan applikasjonene testes direkte. Alle skjermbildene i Figur 6-12 er hentet fra J2WTK.

Figur 6-12 Skjermbilder fra MOBster. Til venstre: En oversikt over filene som ligger lagret på mobiltelefonen. I midten: Hovedmenyen. Funksjonen "Koble til" er merket. Med "Koble til" kan mobiltelefonen koble seg opp mot et relè. Til høyre: Felt for å skrive inn søkestreng.

I enkelte skjermbilder brukes engelske menyvalg. Det er ikke mulig å endre disse. Årsaken til dette er at meny-teksten er avhengig språket som er valgt på emulatoren/telefonen.

6.4 Implementasjon

Med utgangspunkt i designet i avsnitt 6.3 utviklet jeg en prototyp. Det har vært en iterativ prosess, med mye uttesting. Prototypen realiserer P2P fildeling for mobiltelefoner ved hjelp av JXTA for J2ME. Tipsene fra avsnitt 3.7 har blitt tatt hensyn til.

6.4.1 Koden

Kildekoden til MOBster utgjør til sammen 35kB. Det er mulig å gjøre koden enda kortere, men dette har ikke vært noe mål i prosjektet. Koden ligger i sin helhet i vedlegg II. For å kunne kjøre programmet på en mobiltelefon må de kompilerte .class-filene pakkes i en .jar-fil. Siden MOBster bruker JXME sitt API, må også disse .class-filene tas med i .jar-filen. Ferdig pakket er størrelsen på programfilen 26 kB. .jad filen inneholder kun informasjon om versjon, forfatter og plassering av .jar-fil. .jad filen er på rundt 150 byte. En Nokia 6610 har som nevnt tidligere 64 kB minne tilgjengelig til hver enkelt MIDlet suite. Det vil si at mobiltelefonen har omtrent 38 kB ledig minne til lagring av nedlastede filer.

6.4.2 Feil- og unntakshåndtering

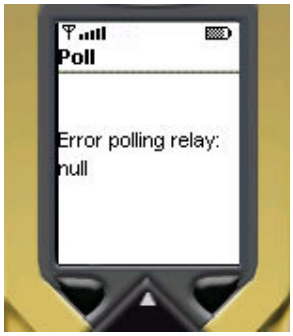
Med en så begrenset skjerm og med så få input-muligheter må informasjonen som vises og de valgene brukeren blir bedt om å ta være enkle og forståelige. Det egner seg ikke med en lang Windows-feilmelding på en mobiltelefon. Program som kjører på mobiltelefoner bør rett og slett unngå å feile. Dette kan høres selvfølgelig ut, men det har både med forventninger fra bruker og begrensningene i mobiltelefonen å gjøre. En mobiltelefon har så langt vært mer en telefon enn en datamaskin. Mobiltelefonen skal kunne ringe, ta i mot samtaler m.m. Brukere forventer at disse funksjonene skal virke feilfritt. På en PC er det annerledes. Der er brukere mer vant til at programmer kan krasje. At disse to plattformene nå konvergerer, skaper utfordringer for utviklere.

Utviklere må ta alle mulige forholdsregler. Forbindelsen kan brytes, *hand-over* mellom basestasjoner kan feile, dekningen blir borte eller en innkommende samtale kan forstyrre programmet.

Som nevnt tidligere i kapittel 2 er det flere måter å håndtere feil på. Tre av disse er:

- Finne feil
- Maskere feil
- Tolerere feil

MOBster er designet med tanke på at den skal kunne finne feil og tolerere feil. Når feil oppstår, for eksempel ved at forbindelsen med JXTA-relè brytes, dukker det opp en melding på skjermen som forklarer at det ikke oppnås kontakt med relè (se Figur 6-13). Beskjeden blir stående til brukeren klikker den bort. Det å finne feil og tolerere feil er med på å gi en viss grad av feiltransparens i applikasjonen.



Figur 6-13 Feilmelding. Mobiltelefonen klarer ikke å polle relèet.

6.4.3 Info-meldinger

De fleste utviklere som har vært borti Java er vant til å lage brukergrensesnitt med J2SE. Brukergrensesnittet kodes for hånd eller designes i utviklingsverktøy som for eksempel Borland JBuilder. Koding av brukergrensesnitt i J2ME krever en annen måte å tenke på. Det finnes et svært begrenset utvalg skjemaer (*forms*). Skjermen er liten og det kan være et problem å få plass til all nødvendig informasjon i den lille ruta. Hjelm [18] kommer med en del nyttige tips når det gjelder å designe trådløse informasjonssystemer (se 6.3.4).

Det er flere kategorier med informasjon som skal presenteres i et display. Feilmeldinger og unntak er diskutert i avsnitt 6.4.2. Menyer og innstillinger er to andre typer informasjon som skal presenteres. Dette presenteres ved hjelp av statiske skjemaer som ligger hard-kodet i programmet (se midterste skjermbilde på Figur 6-12). En tredje kategori med informasjon er dynamisk informasjon som at en fil er lastet ned, størrelsen på en fil og så videre. Denne type informasjon er typisk ikke veldig viktig for at programmet skal fungere, men er der for å gi brukeren informasjon om det som skjer. Informasjonen bør derfor ikke vises lenge, og brukeren bør enkelt og raskt kunne komme tilbake til der han var.

Til dette formålet har J2ME noe som heter **`Alert`** (varsling). En `Alert` er et skjermbilde som viser data til brukeren, og venter en gitt periode før den går videre til neste/forrige skjermbilde.

```
Alert(String title, String alertText, Image alertImage, AlertType alertType)
```

En `Alert` er egentlig laget for å informere brukeren om feil og spesielle tilstander. Men `Alerts` egner seg også godt til å vise vanlig informasjon. Grunnen er at utviklere kan angi en `AlertType`, som er hentet fra `AlertType`-klassen i API-et. Det finnes fem `AlertType`-er.

- *Alarm* – brukes til å gi brukeren beskjed om noe han tidligere har bedt om
- *Confirmation* – ber brukeren bekrefte tidligere valg
- *Error* – gir brukeren beskjed om feil bruk av programmet eller at en feil har oppstått i programmet.
- *Info* – gir ikke-kritisk informasjon til brukeren
- *Warning* – gir brukeren beskjed om potensielle farer eller kostnader ved å uføre en operasjon

AlertType.INFO er flittig brukt i MOBster for å gi informasjon til brukeren (se Figur 6-14).

Figur 6-14 Skjermbilde fra MOBster. Info-meldinger.

6.4.4 Tilkobling

For å kunne kommunisere med JXTA-nettverket er mobiltelefoner som kjører JXME avhengig av å koble seg til et JXTA-relè. Tilkoblingen er kritisk med tanke på funksjonaliteten til programmet. Uten kommunikasjon med relè vil ikke programmet ha stor nytte.

Under valget "*innstillinger*" i menyen (se Figur 6-12) har brukeren selv mulighet til å angi hvilket JXTA-relè han vil bruke. Relèet identifiseres med en IP-adresse og en port (normalt port 9700 i JXTA).

Tiden det tar å koble til relèet kan variere. Årsakene til dette er flere. Belastningen på relèet kan være varierende og hastigheten på nettverket kan variere. Under testing har vi sett at det er stor forskjell på hvor lang tid en JXME-peer bruker på å koble seg til forskjellige versjoner av *JXTA-shell* som fungerer som relè. To forskjellige versjoner ble testet i prosjektet, med svært varierende resultat. Det ble registrert en oppkoblingstid på alt i fra 1 sekund til over 15 sekunder. En årsak til dette kan være at teknologien fortsatt er umoden.

For å gi brukeren informasjon om status på tilkobling, benyttes en subklasse `StatusUpdate` som arver fra klassen `TimerTask` i J2ME. Denne tilbyr en grafisk fremstilling av tiden som er gått siden brukeren ba om tilkobling (se Figur 6-15).

Figur 6-15 Skjermbilde fra MOBster. Tilkobling til relè.

6.4.5 Søk

Søking er essensielt i en fildelingsapplikasjon. Algoritmer som benyttes i søkefunksjonene har mye å si for ytelse og hvor gode resultater som oppnås (rekkevidde og antall treff). Hvilke fremgangsmåter en velger baseres på hva slags teknologi som ligger til grunn for applikasjonen. Som nevnt i kapittel 2 deler vi P2P-fildelingssystemer inn i to kategorier, **ren** P2P og **hybrid** P2P.

Metoder som brukes i hybride P2P-systemer, som for eksempel Napster, egner seg dårlig i mobile omgivelser. Det er to grunner til dette sier Lindemann og Waldhorst [6]. For det første finnes det sjelden terminaler i et trådløst nett som GSM, som er kraftige nok og har stor nok lagringskapasitet til å tilby en slik sentralisert indekseringstjeneste. En slik tjeneste måtte derfor ha blitt plassert på en PC som kunne nås fra terminalene i et trådløst nett. For det andre er forbindelsene i mobile nettverk mer ustabile på grunn av terminalenes mobile natur. Det er ingen garanti for at alle terminaler tilknyttet nettet alltid vil kunne nå en sentralisert terminal.

Metoder som brukes i rene P2P-systemer egner seg bedre i mobile omgivelser. Årsaken er at feiltoleransen er større. Selv om enkelte peer-er mister forbindelsen til nettverket eller kobler seg av med vilje, vil ikke dette påvirke søket nevneverdig. Andre peer-er vil kunne svare på forespørslene og rute søket videre.

JXME tilbyr en slags hybrid P2P-modell fordi mobiltelefonene er avhengig av et relè. Forskjellen fra typiske hybrid-modeller er at brukeren selv kan velge relè. Selve søkene utføres på samme måte som i rene P2P-systemer. Metoder for å velge relè dynamisk er i følge [59] under utvikling. En JXME-peer tilknyttet et relè kan dessuten kommunisere med en JXME-peer tilknyttet et annet JXTA-relè et annet sted i nettet.

Det finnes forskjellige måter å løse søking i mobile P2P-systemer på. Lindemann og Waldhorst [6] har foreslått en metode kalt *Passive Distributed Indexing* (PDI) som en tilnærming til fildeling i mobile omgivelser. PDI definerer et meldingsformat som består av en rekke felt. Et av dem er TTL (Time To Live)-feltet som brukes for å begrense søkene (hindre at søket oversvømmer nettverket). Det defineres en *document identifier* som gir hvert eneste dokument i nettverket en unik adresse bygd opp av dokumentets lokale adresse og adressen til terminalen (IP-adresse, MAC-adresse eller URI). I tillegg introduseres en slags indeks-*cache*[30] i hver enkelt terminal. Denne *cache*-en inneholder stikkord for hvert enkelt dokument. Dette gjør søkingen effektiv.

MOBster
Søkefunksjonen i MOBster tar utgangspunkt i de grunnleggende funksjonene i JXTA og JXME. Ved hjelp av en forhåndsdefinert pipe, MOBster pipeID (se Figur 6-8), sendes alle forespørsler rundt til alle peer-er som lytter på denne bestemte pipen. URI-en til pipen ligger hardkodet i programmet for at alle peer-er skal få tilgang til den. Pipen er av type *propagate* (utbredelse). Det vil si at alle meldinger som sendes inn i den mottas av alle som lytter på den. Etter at en forespørsel er mottatt og behandlet sendes responsen via en *unicast* (ende-til-ende) pipe tilbake til avsender.

Alle meldinger som sendes er bygget opp av et forhåndsbestemt meldingsformat som alle MOBster-peer-er forstår. I meldingen defineres det hva slags type melding det er. Kommandoen for søk er *searchString*. Mer om meldingsformatet i avsnitt 6.4.7.

Lokalt søk
Når en peer mottar en melding fra en annen peer, analyseres denne med *parseCommand*(). Parseren tolker meldingen og finner ut hva som skal gjøres med den (se Figur 6-10). Ved oppstart av MOBster leses alle navnene til lokalt lagrede filer inn i en **List**[31] fra RMS (Record Management System). Denne listen fungerer som en slags cache. Programmet trenger ikke å kontakte RMS hver gang den skal søke etter en fil. Listen gjør filnavnene enkelt tilgjengelig når peer-en mottar forespørsler. Dette gjøres for å begrense CPU-bruken og for at programmet skal ha rask aksess til filnavnene. Det er viktig å begrense strømforbruket og tidsforbruket.

Når *parseCommand* har funnet ut at den innkommende meldingen inneholder et søk, kalles en metode for å søke etter filnavnet i listen over filer. Hvis filen eksisterer sendes det en respons tilbake til avsender med melding om at denne filen finnes. Hvis filen ikke eksisterer sendes det ingen respons. Dette diskuteres nærmere i avsnitt 7.4.3.

[30] Cache = hurtigbuffer
[31] List – en ferdiglaget struktur i J2ME

Foreløpig er det kun mulig å søke på filnavn. I en eventuelt ny versjon av programmet må dette forbedres. En måte å gjøre det på kan være ved å benytte samme cache-funksjon som brukes i PDI. Metadata i form av stikkord beskriver hver enkelt fil. Søkene foretas i disse stikkordene. Det bør også være mulig å søke med såkalte "*wildcards*" som '*' og '?', eller deler av filnavn.

6.4.6 Polling

Etter å ha koblet seg til JXTA-relèet går MOBster over i en evig løkke som poller relèet med et fast intervall. Fordi J2ME (MIDP 1.0) kun har støtte for utgående HTTP er dette eneste måten å få hentet ned meldinger og data til mobiltelefonen fra JXTA-nettverket på. Med MIDP 2.0 vil det bli mulig å bruke noe som heter "*Push registry*". Med *Push Registry* vil det ikke være behov for å polle etter innkommende meldinger. Meldingene kan i stedet "*pushes*" direkte til mobiltelefonen.

Pollingen krever ressurser både på mobiltelefon og på relè. Under "innstillinger" i menyen er det mulig å velge hvor ofte mobiltelefonen skal polle relèet. Nedad er intervallet begrenset til minimum ett sekund. Oppad er det ingen begrensning, men vi anbefaler å ikke velge noe særlig mer enn fem sekunder. Årsaken til dette er at responstiden vil bli for lang til at applikasjonen egner seg til normal bruk. Størrelsen på intervallet er en avveining mellom effektivitet og kostnader. Med GPRS betaler brukeren for hvor mye trafikk som sendes og mottas. Det er ikke store datamengdene som overføres i forbindelse med ett poll, men over tid kan pollingen utgjøre i overkant av 80% av den totale trafikken. Mer om dette i avsnitt 6.5.3.2.

Pollingen fører til at mobiltelefonen får en respons i fra relèet. Det er en rekke forskjellige meldinger som kan komme. For enkelhets skyld er det greit å dele disse inn i to kategorier (se også 5.3.2):

- **JXME-spesifikke**: Meldinger som relèet genererer i forbindelse med oppretting av piper, peer-er og grupper og utsending av meldinger. Det er fem type meldinger: *Success*, *Info*, *Error*, *Result*, *Message*. Disse ble presentert i avsnitt 5.3.2.
- **Program-spesifikke**: Meldinger som blir generert av mobiltelefonene. Bygd opp på meldingsformatet laget for MOBster. Mer om dette i avsnitt 6.4.7.

6.4.7 Sending av meldinger

"Et distribuert system er et system der komponenter lokalisert på datamaskiner, koblet sammen i et nettverk, kommuniserer og koordinerer sine handlinger bare ved å utveksle meldinger". Slik definerer [16] et distribuert system. Meldingene er hjertet i hele systemet. Figur 6-16 viser tre forskjellige metoder jeg utviklet for å sende meldinger i MOBster:

sendSearch(String searchString)	Sender søk (searchString) ut på en kjent pipe som alle MOBster-applikasjoner lytter på.
sendData(String filename, String reciever, String senderPipe)	Sender en gitt fil filename til en mottaker reciever som lytter på en pipe med adresse senderpipe.
sendResult (String msg, String reciever, String senderPipe)	Sender et svar msg tilbake til en mottaker reciever som lytter på en pipe med adresse senderpipe.

Figur 6-16 Metoder for å sende meldinger

Meldingsformat

For å realisere fildeling mellom mobiltelefoner ble det nødvendig å utvikle et eget meldingsformat for MOBster. JXME danner kun et rammeverk for utvikling av P2P-applikasjoner. Applikasjonsspesifikke funksjoner må utviklere lage selv. Jeg utviklet derfor en enkel **protokoll** som bruker meldinger til å utveksle informasjon mellom peer-ene. Meldingsformatet er enkelt og utvidbart. Alle meldinger, *Message*, inneholder en tabell med *Element*-er (se Eksempel 6-1). Kommando-elementet inneholder protokoll-kommandoer som gjør det mulig for mobiltelefonene å dele filer og laste ned filer fra hverandre.

```
Element[] elm = new Element[3];
Message m = new Message(elm);
```

Eksempel 6-1 En tabell med tre Elementer i en Message.

Figur 6-17 til Figur 6-19 viser hvordan meldingene blir bygget opp:

sendSearch

Figur 6-17 Meldingsformat for søk

SendSearch er bygget opp av tre elementer. Navnet på avsender, en kommando og avsenders adresse. Det er foreløpig bare laget en kommando i forbindelse med søk:

- **searchString** - søkestrengen

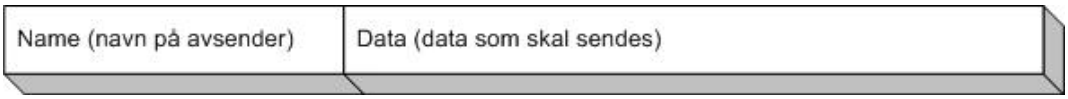
sendResult

Figur 6-18 Meldingsformat for *response*

Et riktigere navn på denne metoden hadde vært *sendResponse*. I virkeligheten er det ikke bare resultater som sendes med denne metoden. Det sendes også feilmeldinger og forespørsler om nedlasting av filer. Meldingen er bygget opp på samme måte som ved et søk. Det finnes tre typer kommandoer:

- **result** – resultat av søk
- **get** – ber om at valgt fil skal lastes ned
- **error** - feilmelding

sendData



Figur 6-19 Meldingsformat for utsending av data

SendData har kun to elementer. Et med avsenders navn og et med data. Lengden på data-feltet er variabel.

6.4.8 Filbehandling

En stor del av koden som er laget i forbindelse med MOBster tar for seg filbehandling. I avsnitt 3.3.3 så vi på RMS og mulighetene for persistent lagring av data på mobiltelefonen. Funksjonene og mulighetene i RMS er svært begrenset med tanke på lagring av filer. Med filer mener vi bilder, video og lyd i form av en *byte-array* (byte[]). Ren tekst er enklere å lagre i RMS. Ferdige metoder som

```
writeUTF(String s);
```

hentet fra `DataOutputStream`-klassen under `java.io`, gjør det mulig å skrive hele strenger direkte inn i RMS. Det henvises til [57] for mer informasjon om RMS.

Hvis du laster ned en fil, er det som regel fordi du ønsker å bruke denne filen. Enten det er et bilde, en lydfil, en videosnutt eller en ny ringetone er det ønskelig å kunne bruke og teste ut filene. Med .png (Portable Network Graphics)-bilder er ikke dette noe problem. J2ME har metoder for å vise frem slike bilder på skjermen. For andre filtyper er det verre. Det finnes ingen metoder i J2ME for å vise frem .jpg eller .gif bilder direkte. I MOBster er det derfor kun støtte for .png-bilder.

I J2ME er det kun programmer innenfor samme MIDlet suite som har tilgang til informasjonen som er lagret i RMS. For å kunne se på .jpg-bilder eller se på en .mpg videosnutt måtte en egen media-avspiller bygget på MMAPI (Multimedia-API-et) [57] blitt integrert i samme MIDlet suite som MOBster. Med dagens begrensninger på maks 64 kB per MIDlet suite er ikke dette mulig.

I fremtiden ser vi for oss at det blir mer minne tilgjengelig for hver enkelt MIDlet. Da vil det være mulig å lage en komplett applikasjon bestående av fildeling, media-avspiller og chat.

På grunn av begrensninger med å lagre filer (.png-bilder) direkte i RMS, har det vært nødvendig å utvikle metoder for å ta hånd om bilde-filene. Utviklingen har vært konsentrert rundt de mest fundamentale funksjoner ved filbehandling:

- Sletting
- Lagring

Ved oppstart av programmet leses alle navnene til lokalt lagrede filer inn i en liste ved hjelp av *readFiles*()-metoden. Alle filer lagres ved hjelp av *saveFiles*()-metoden. *SaveFiles*() kalles automatisk ved mottak av nye filer, ved sletting av filer og når brukeren går ut av skjermbildet ”Mine filer”.

Mine filer
Under menyvalget ”mine filer” får brukeren opp en oversikt over alle filer som er lagret på mobiltelefonen. Ved hjelp av menyen har brukeren tilgang til følgende:

- **Info** – gir brukeren informasjon om filstørrelse og filtype.
- **Endre** – vil i en eventuell senere versjon av programmet gi brukeren mulighet til å endre filnavnet.
- **Slett** – sletter valgt fil.
- **Tilbake** – tar brukeren tilbake til hovedmenyen.

Nye filer som lastes ned dukker automatisk opp i denne listen. Det er forløpig ikke mulig å velge hvilke filer som skal deles og ikke. Det vil si at alle filer som befinner seg på mobiltelefonen blir delt. I en eventuell ny versjon av MOBster bør det lages funksjoner som lar brukeren velge hvilke filer som skal deles og hvilke som ikke skal deles.

Filstørrelser
De første .png bildene som ble brukt under testingen av programmet hadde en filstørrelse på rundt 250 Byte. Dette var små bilder på 20x20 piksler. I en virkelig applikasjon vil behovet for å laste ned større bilder melde seg. Under uttestingen ble det derfor testet med filer helt opp til 3500 byte. Dessverre oppstod det feil under sending av disse. Det vil si, bildene ble sendt fra avsender, men de kom aldri frem til mottaker.

I følge JXTA Community er øvre grense på filer rundt 10MB. At pakker blir borte, forklares med at JXTA ikke har noen garanti for at pakker kommer frem.

6.5 Testing

Testing har vært viktig gjennom hele utviklingsprosessen. Fordi teknologien er såpass umoden og det forelå veldig lite dokumentasjon, ble mye av kunnskapen tilegnet gjennom prøving og feiling.

Metoden som er benyttet i prosjektet er protoyping. Det har aldri vært en hensikt å lage et ferdig produkt. Prototypen eller demonstratoren skal brukes som et eksempel på hvordan P2P fildeling på mobiltelefoner kan realiseres. Den skal være med på å vise at P2P fildeling lar seg implementere på mobiltelefoner, samt gi erfaringer med hensyn til hvor moden dagens teknologi er.

6.5.1 Testing på emulator

Testingen har foregått på emulatorer i J2ME Wireless Toolkit (J2WTK). Emulatorer gjør det enkelt å teste programmene i tillegg til at de gir et godt inntrykk av hvordan brukergrensesnittet fungerer. Funksjoner for å kompilere ligger ferdig i J2WTK. Det er forskjellige meninger om hvorvidt emulatorene gir et godt bilde av hvordan applikasjonen vil oppføre seg i virkeligheten. Til tross for at emulatorene jobber i et miljø som tilsynelatende er likt det som er på mobiltelefonen, vil det likevel være forskjeller. Eksempler på dette er hastighet på nettverkstilknytning, forsinkelse, prosessorkraft og minne. En annen stor ulempe som ikke nødvendigvis bare emulatorene har skyld i, er at de forskjellige mobiltelefonleverandørene har forskjellige implementasjoner av CLDC og MIDP. Dette gjør at et program som fungerer fint i emulatoren ikke nødvendigvis fungerer like bra på en mobiltelefon. Dette er noe vi har fått erfare i dette prosjektet.

J2WTK har funksjoner for å monitorere minneforbruk og nettverkstrafikk. Spesielt funksjonen for å monitorere nettverkstrafikk har vært nyttig i prosjektet. Den gjør det mulig å se alt innhold i alle pakker som beveger seg inn eller ut fra mobiltelefonemulatoren.

En viktig del av testingen har bestått i å teste hvordan de forskjellige peer-ene oppfører seg overfor hverandre, det vil si hvordan de interagerer. Selv om et program kjører feilfritt alene, er det ikke sikkert dette er tilfelle når flere peer-er kobler seg til nettverket. Det er flere eksempler på dette fra utviklingsarbeidet. I ett tilfelle fikk alle peer-er den samme URI-en tildelt. Programmet fungerte fint når det kjørte alene, men når flere peer-er kom inn i bildet ble det helt kaos. Meldinger ble levert til feil mottaker, og ingen ting fungerte.

Noe vi også fikk erfare er at det ikke er mulig å kjøre to peer-er på samme maskin med J2ME Wireless Toolkit. Problemet er at begge programmene vil lese og skrive til samme plass i RMS. Alle *Record Stores* (databasefil) har et navn. Siden implementasjonene er like, er også navnet på databasefilene like. Når de to peer-ene forsøker å skrive til eller lese fra RMS, blir det kaos. Dermed vil det som tilsynelatende er to peer-er, egentlig oppføre seg som en peer. Emulatorene må derfor kjøre på hver sin maskin for å skape et testmiljø som fungerer.

For at JXME-peer-er skal kunne kommunisere med omverdenen må de koble seg opp mot et JXTA-relè. Vi har testet med oppkobling både mot relè som kjører lokalt på den maskinen der utviklingen har skjedd og med JXTA Community sitt relè. Det har også blitt gjort tester der de forskjellige peer-ene kobler seg opp mot forskjellige relèer. Resultatet fra denne testingen har vært varierende. Det virker som det er helt vilkårlig hvilke meldinger som kommer frem og hvilke som ikke gjør det. En e-post angående problematikken ble sendt til JXTA Community. *Akhil Arora*, en av de tre lederne for JXME-prosjektet ga til svar at årsaken til dette er ”*the nature of JXTA propagate pipes*”. Ingen ordnet levering eller pålitelighet er garantert. Tap av meldinger kan forekomme ved stor trafikk i nettet eller når enkelte peer-er blir utilgjengelige. For å kunne tilby QoS (Quality of Service), må utvikleren i dag lage egne protokoller over JXTA som garanterer at meldingene kommer frem. Det vil bli implementert pålitelige piper i senere versjoner av JXTA.

6.5.2 Testing på mobiltelefon

Neste trinn i testingen var å teste applikasjonen på mobiltelefoner. En god del av dagens mobiltelefoner har støtte for Java. Det har derfor hele tiden vært et mål å få testet applikasjonen på mobiltelefoner. En ting er å bruke programmet i en emulator på sin egen datamaskin, men det blir noe helt annet når brukeren kan holde mobiltelefonen i hånda og se hvordan programmet oppfører seg i sine virkelige omgivelser.

Det viste seg å bli vanskeligere enn ventet å få testet programmet på en mobiltelefon. Vi hadde to testtelefoner tilgjengelige. Den ene var en Nokia 9210 Communicator, som kjører en MIDP-klient. Den andre var en Nokia 3510i (se Figur 3-4). Mens Nokia 9210 Communicator har en seriellkabel som kan brukes til å overføre filer til og fra telefonen, er den eneste muligheten til å få filer/programmer inn på Nokia 3510i ved hjelp av OTA (se avsnitt 3.5). Dette gjorde at vi var avhengig av en webtjener med en statisk IP-adresse som kunne nås fra en mobiltelefon.

I kapittelet "Testing på emulator" ble det nevnt at ikke alle implementasjoner av CLDC og MIDP er helt like. Dette fikk vi erfare da vi tok .jad og .jar filene som J2WTK hadde generert, og overførte disse til mobiltelefonene. Nokia 9210 Communicator nektet å starte programmet. Nokia 3510i nektet å ta i mot programfilene i det hele tatt. Det er tydelig at Sun og Nokia ikke er helt enige om hvordan J2ME bør implementeres.

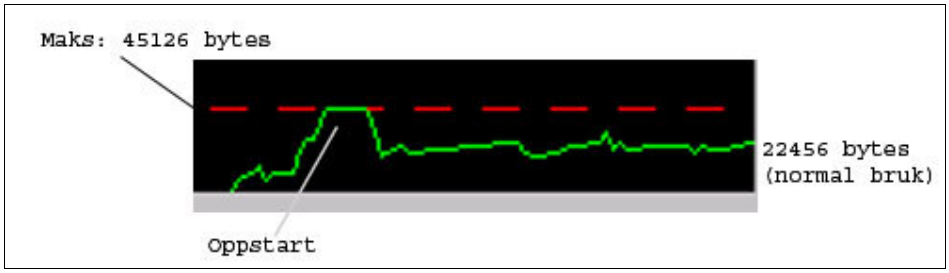
Løsningen ble å laste ned Nokia sin "Developer's Suite for the Java 2 Platform, Micro Edition v 1.1" [54] samt en Nokia 3510i-modul til denne. Når .class-filene ble pakket med Nokia sin Developer's Suite, godtok mobiltelefonene programfilene. Dessverre fikk jeg problemer når mobiltelefonene skulle koble seg til relèet. Hver gang mobiltelefonene forsøkte å polle, kom det feilmeldinger. For å være sikker på at dette ikke skyldtes programmet, testet jeg også Chat-eksempelet, utviklet av JXTA Community. Chat-programmet genererte akkurat de samme feilmeldingene når det koblet seg opp mot relèet. Disse resultatene ble lagt frem for JXTA Community, uten at det var til særlig hjelp. Det ser ut til at det kan være en konflikt mellom Nokia sin implementasjon av CLDC og MIDP og JXME. Dette er et tegn på at teknologien foreløpig er umoden.

6.5.3 Ytelsestesting

Som nevnt over har det ikke vært mulig å teste applikasjonen på mobiltelefoner. Det er derfor vanskelig å si hvordan applikasjonen ville ha fungert under normal bruk i sine virkelige omgivelser. Ved hjelp av J2WTK har vi mulighet til å danne oss et bilde av hvordan det vil fungere. Vi har ved hjelp av J2WTK undersøkt minneforbruket (avsnitt 6.5.3.1) under kjøring av applikasjonen samt nettverkstrafikken (avsnitt 6.5.3.2) som går til og fra telefonen. I avsnittet 6.5.3.3 presenteres resultatene fra en test som undersøker hvor lang tid det tar å få svar på forespørsler sendt ut i nettverket. Fordi dette kun er gjort på emulatorer, vet vi lite om hvordan dette ville artet seg i virkeligheten. Høyst sannsynlig ville det gå enda tregere over mobilnettet.

6.5.3.1 Minneforbruk

Målet med applikasjonen har vært å realisere P2P fildeling for mobiltelefoner. Effektivitet og minneforbruk har ikke stått i fokus. Likevel var det interessant å se på minneforbruket i programmet. Testingen ga ganske oppsiktsvekkende resultater. På forhånd var det ventet at forbruket av minne skulle være størst når bruken av programmet var på det mest intense, for eksempel ved utsendelse av søk, eller nedlastning av filer. Det viste seg imidlertid at forbruket av minne var aller størst ved oppstart av programmet.

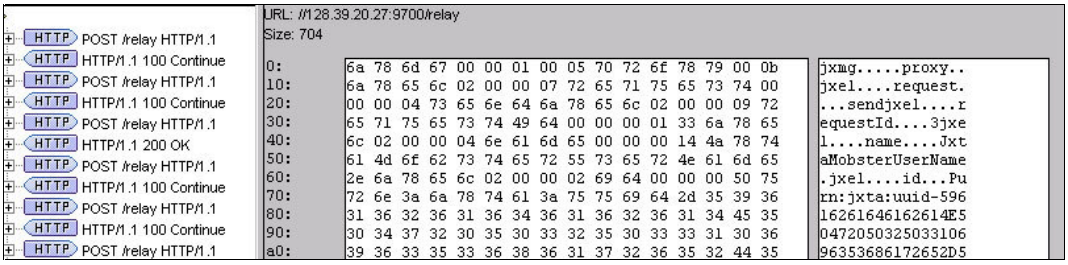


Figur 6-20 minneforbruk i mobster (grafen er hentet fra J2ME Wireless Toolkit)

Som Figur 6-20 viser er minneforbruket nesten dobbelt så stort ved oppstart som ved normal bruk. Årsaken til dette er det vanskelig å fastslå. Da vi utførte liknende tester med programmer som ikke benyttet JXME eller kommuniserte over HTTP var forskjellen mellom maksimalt minneforbruk og normal veldig liten. Det er derfor trolig JXME som krever en del ressurser under initialiseringen. Det er viktig å være klar over denne toppen i minneforbruk som kan oppstå ved bruk av JXME. Mange mobiltelefoner vil kunne ha problemer med å takle et så høyt minneforbruk.

6.5.3.2 Nettverkstrafikk

Siden båndbredden som er tilgjengelig i GSM- og GPRS-nettet er veldig begrenset og kostbar, er det viktig å ha kontroll med hvor mye trafikk som sendes til og fra mobiltelefonen. Med monitoreringsfunksjonen som finnes i J2WTK kunne dette sjekkes på en enkel måte. All trafikk som går til og fra blir logget. Det er mulig å se alt innholdet i pakkene som sendes, i tillegg til størrelsen på disse (se Figur 6-21).



Figur 6-21 oversikt over nettverkstrafikk (hentet fra J2ME Wireless Toolkit)

Fordi JXME benytter MIDP 1.0, som kun har støtte for utgående HTTP, var vi klar over at applikasjonen ville generere mer nettverkstrafikk enn det som er helt nødvendig. Pollingen mot relèet genererte en jevn strøm av trafikk, som vi valgte å kalle **tomgangstrafikk**. Hvor stor denne tomgangstrafikken var, og hvor stor del av den totale trafikken denne utgjorde visste vi lite om.

For å få et så realistisk resultat som mulig, satte vi opp et nettverk bestående av fire peer-er. Alle peer-er var koblet til samme relè. Hver peer var tilkoblet nettverket i 15 minutter. I løpet av dette kvarteret etterspurte hver peer tre bilder, som de også lastet ned. Det ble også lastet opp tre bilder fra hver enkelt peer. Filene/bildene som ble brukt var på rundt 200 byte. Intervallet for hvor ofte en mobiltelefon poller releèt kan bestemmes av brukeren selv. Størrelsen på intervallet blir en avveiing mellom pris og effektivitet. Vi testet med to forskjellige intervaller. Det minste var polling hvert andre sekund, det største hvert femte sekund. Som nevnt tidligere egner større intervall enn fem sekunder seg dårlig for vanlig bruk. Tabell 6-2 og Tabell 6-3 viser en oversikt over resultatene fra testen:

Polling hvert femte sekund

	Sending	Mottak	% av alt
Oppstart	870	1712	2,32
Søk	5742	20865	23,81
Behandling av søk	4488	10719	13,60
Tomgang i 14 min	0	67368	60,27
Sum	**11100**	**100664**	**100**

Tabell 6-2 Polling hvert femte sekund (størrelser er i Bytes). Tabellen viser hvor mange bytes som sendes og mottas i forbindelse med de forskjellige aktivitetene. Høyre kolonne viser hvor mange prosent aktiviteten utgjør av totalen.

Polling hvert andre sekund

	Sending	Mottak	% av alt
Oppstart	870	1712	1,15
Søk	5742	20865	11,84
Behandling av søk	4488	10719	6,76
Tomgang i 14 min	0	180450	80,25
Sum	**11100**	**213746**	**100**

Tabell 6-3 Polling hvert andre sekund (størrelser er i Bytes). Tabellen viser hvor mange bytes som sendes og mottas i forbindelse med de forskjellige aktivitetene. Høyre kolonne viser hvor mange prosent aktiviteten utgjør av totalen.

Det mest oppsiktsvekkende ved dette resultatet er hvor stor del av totaltrafikken tomgangstrafikken utgjør. Med polling hvert femte sekund utgjør tomgangstrafikken i overkant av 60% av totaltrafikken. Med polling hvert andre sekund utgjør tomgangstrafikken hele 80,25%. Hvilke konsekvenser dette har med tanke på kostnader vil bli diskutert senere i rapporten.

6.5.3.3 Tidsforbruk

En bruker kan ha mange krav til en fildelingsapplikasjon. Den skal være enkel å bruke, den skal være billig i drift og så videre. Likevel er noe av det mest kritiske for en slik applikasjon tidsforbruk. Hvis det tar 3-4 minutter å utføre et søk vil ingen orke å bruke programmet. Det tar rett og slett for lang tid. Med utgangspunkt i testkonfigurasjonen beskrevet i avsnitt 6.5.3.2 undersøkte vi hvor lang tid det tok å utføre søk og laste ned filer.

Testing på emulatorer foregikk under optimale forhold, med maskiner knyttet sammen i et lokalnett[32]. På mobiltelefoner ville vi nok opplevd litt andre responstider (se Tabell 3-3, side 36). Det var kun fire emulatorer tilkoblet nettverket. Alle emulatorer var koblet opp mot samme relè. Det ble benyttet polling hvert eneste sekund. Tabell 6-4 viser en oversikt over resultatet:

Tidsforbruk

	Snitt	Min	Maks
Tid til første svar (s)	6,5	5,4	9,0
Tid til siste svar (s)	10,3	8,9	13,5
Tid å laste ned (s)	8,7	5,9	12,1

Tabell 6-4 Tidsforbruk ved søk (i sekunder)

[32] Ethernet 100Mbit/s (IEEE 802.3u)

Disse resultatene er innhentet under optimale forhold. Hadde vi lagt til ekstra forsinkelse på bakgrunn av Tabell 3-3, side 36 ville resultatene sett annerledes ut. Forsinkelsen og den begrensede båndbredden på mobiltelefoner ville nok gjort store utslag. Dessuten er nok et polleintervall på ett sekund i overkant av hva en mobiltelefon ville klart i virkeligheten. Pollingen er kritisk, siden det kun kan hentes inn en innkommende melding av gangen.

Dette gir likevel et slags bilde av hvor lang tid det tar å utføre søk i et nettverk. Tid fra første svar til siste svar sammen med polleintervallet, kan hjelpe oss å si noe om størrelsen på nettverket. I eksempelet over er differansen mellom første og siste respons 3,8 sekunder. Med et polleintervall på 1 sekund sier det oss at nettverket består av omtrent tre andre peer-er utenom en selv. Hvor lang tid det tar før mobiltelefonen får første respons avhenger også av hvor ofte de andre peer-ene i nettverket poller relèet. Hadde peer-ene vært koblet til forskjellige relèer ville forsinkelsen vært enda større.

Ut i fra resultatene over er det mulig å beregne hvor lang tid det vil ta mellom første og siste respons i et stort nettverk. Tiden brukeren må vente på siste svar (t_{ss}) vil være gitt ved:

$$t_{ss} = (t_i * n) + t_f$$

der t_i er polleintervallet, n er antallet peer-er og t_f er tiden til første svar kommer. Allerede med 20-30 peer-er i nettverket vil det ta for lang tid å få tilbake svar. Skalerbarheten til systemet diskuteres i neste kapittel.

6.6 Oppsummering

I dette kapittelet har vi sett på utviklingen av prototypen som danner grunnlaget for denne rapporten. Først så vi på mulige scenarier i forbindelse med P2P fildeling for mobiltelefoner. Et scenario så på muligheten av å la mobiltelefonen være en utstrakt arm (fjernkontroll) for hjemme-PC-en. Det andre scenariet tok for seg fildeling direkte mellom mobiltelefoner ved hjelp av JXTA relè. Et tredje scenario tok for seg muligheten av å la mobiltelefoner ta del i fildeling mellom PC-er ved hjelp av JXTA. Det siste scenariet så på muligheten av å overføre filer via BlueTooth. Fokuset i oppgaven har vært på scenario to, avsnitt 6.1.2, som beskriver fildeling bare mellom mobiltelefoner.

Videre beskrev kapittelet de neste stadiene i utviklingsprosessen, fra kravspesifikasjon og design til implementasjon av programmet. Kravspesifikasjonen er i sin helhet lagt ved oppgaven som et vedlegg. Design-avsnittet beskrev systemet i form av en arkitekturmodell, et klassediagram og forskjellige sekvensdiagrammer. I avsnittet om implementasjonen tok vi for oss brukergrensesnitt, valg av søkemetoder, format på meldinger og filbehandling.

Til slutt tok vi for oss testing av systemet. Det har blitt gjort tester på minneforbruk og nettverkstrafikk. Vi innførte et begrep, **tomgangstrafikk**, som er all trafikk som blir generert i forbindelse med polling. Vi så også på responstider og hvordan dette påvirker systemet.

I neste kapittel, Diskusjon, vil prototypen bli diskutert nærmere.

7 Diskusjon

I de foregående kapitlene så vi på prototypen og de teknologier som ligger til grunn for den. Dette kapittelet tar for seg diskusjon rundt P2P fildeling for mobiltelefoner og resultater fra testene gjort på prototypen. Etter diskusjonen sammenfattes oppgaven i en konklusjon i kapittel 8. Hva er egentlig så spesielt med P2P-fildeling for mobiltelefoner? Er fildeling på mobiltelefoner noe nytt? Er fildeling lovlig/ulovlig? Hvilke forbedringer kan gjøres på applikasjonen/JXME for å få det til å fungere bedre?

Disse spørsmålene skal vi prøve å besvare i avsnittene under. Kapittelet er delt opp i avsnitt som diskuterer P2P-fildeling (avsnitt 7.1), JXTA i forhold til andre teknologier (avsnitt 7.2), JXME (avsnitt 7.3) og prototypen (avsnitt 7.4). Til slutt diskuteres mobilitet i forhold til prototypen og relatert arbeide.

7.1 P2P

I Internetts spede begynnelse på slutten av 60-tallet, var nettet et Peer-to-Peer system. Målet med det opprinnelige ARPANET [26] var å dele prosessorkraft mellom universiteter i USA. Da Internett fikk sitt gjennombrudd på begynnelsen av 90-tallet, var hastigheten på forbindelsene inn mot Internett liten. De fleste koblet seg opp med analoge modem. Kraftige tjenere behandlet forespørsler på vegne av klienter rundt i hjemmene. Fra å være ren P2P i begynnelsen, ble Internett mer og mer sentralisert. Til slutt ble det en ren klient/tjener arkitektur [26], der pc-ene bare sendte forespørsler og mottok svar fra webtjenerne.

I dag er situasjonen en helt annen. Flere og flere får bredbånd og PC-ene blir kraftigere og kraftigere. Til tross for at vi nå får mer båndbredde, er ikke nettverkene spesielt godt egnet til P2P-trafikk. Mange brukere har asymmetriske Internettforbindelser, for eksempel ADSL (Asymmetric Digital Subcriber Line), med vesentlig høyere hastighet i nedlastning enn opplastning. Telenor Internett sin billigste ADSL-pakke[33] kan tilby 704 kbit/s i nedlastning og 128 kbit/s i opplastning. Mengden trafikk du kan sende og motta er begrenset til 1GB i måneden. Dette egner seg dårlig til å kjøre P2P-trafikk, der det som regel er ønskelig å kunne kjøre samme hastighet ut og inn om alle skal bli behandlet som likeverdige. Begrensninger i mengde trafikk som kan overføres, er også et hinder for P2P-trafikk.

Nå som mobiltelefonene har fått muligheten til å koble seg opp mot Internett og kjøre egne applikasjoner, er det noe vi kan lære av historien om Internett for stasjonære maskiner? Mobiltelefoner er inne i den fasen som stasjonære PC-er var på midten av 90-tallet. Nettverksforbindelsene har liten båndbredde og mobiltelefonene gjør veldig lite annet enn å sende noen forespørsler og ta i mot svar fra webtjenere. Begrensningene i antall IP-adresser med IPv4 gjør at nettverksoperatørene tildeler adresser dynamisk. Dette gjør at det er vanskelig å få til direktekommunikasjon mellom mobiltelefoner. Klient/tjener er derfor den mest brukte arkitekturen for dagens applikasjoner på mobiltelefoner. Vi ser likevel en tendens til at dette er i ferd med å snu seg. Flere mobiltelefoner kommer nå med IM (Instant Messaging) installert på telefonen. IM lar to brukere skrive meldinger til hverandre, Peer-to-Peer, over GSM eller GPRS.

P2P er ikke bare en måte å desentralisere funksjonalitet og tjenester på. Det er også en måte å desentralisere kostnader og administrasjon på [7]. Sentraliserte systemer er ressurskrevende for de som drifter systemet. Nettverksforbindelser med stor båndbredde, spesialtilpasset maskinvare og programvare er kostbart. Disse utgiftene må dekkes inn på en eller annen måte. I P2P-systemer er kostnadene og administrasjonen fordelt mellom peer-ene i nettverket.

[33] Online ADSL. 995,- i etablering og 345,- i måneden. Maks 1GB med trafikk i måneden. (Per februar/mars 2003)

Kan en mobiltelefon og en PC opptre som likeverdige?
Peer-to-Peer handler om at deltakere i et nettverk opptrer som likeverdige ovenfor hverandre uten innblanding fra sentraliserte terminaler. Ved hjelp av JXTA for J2ME har mobiltelefoner muligheten til å delta i et P2P-nettverk med PC-er, PDA-er og andre terminaler med nettverkstilknytning. Hvor "lik" er egentlig en PC og en mobiltelefon? Kan vi snakke om disse som likeverdige parter?

I følge teorien kan en mobiltelefon i JXTA-verdenen, ved hjelp av relèet, bli så likeverdig en PC som det går an. Mobiltelefonen kan delta i JXTA-nettverket på lik linje med andre peer-er. Denne oppgaven har vist at dette kan være problematisk i praksis med hensyn til fildeling. Til tross for alle begrensninger i dagens telefoner, vil mobiltelefoner etterhvert som de utvikler seg og blir mer ressurssterke kunne opptre mer og mer på lik linje med en PC.

7.1.1 Fildeling
Fildeling vil si at en bruker av et system skal kunne:
- gjøre egne filer tilgjengelig for andre brukere
- søke etter filer hos andre brukere
- laste ned filer fra andre brukere

I bedrifter eller organisasjoner er det vanlig at de ansatte har tilgang til en del felles informasjon. De ansatte kan endre på informasjonen/filene og legge til flere filer. Filene er som regel plassert på en sentral maskin. Dette er fildeling i enkleste forstand. En gruppe mennesker som deler noen felles ressurser/filer. Med P2P fildeling deles all informasjon direkte mellom peer-ene i nettverket. Informasjonen til den enkelte peer gjøres tilgjengelig for andre peer-er. Peer-ene kan søke og laste ned informasjon fra hverandre og ikke fra en sentralisert maskin.

Det er ikke alltid P2P er beste løsning. Noen ganger kan en balansegang mellom P2P og klient/tjener være den beste løsningen. Et eksempel på dette er systemer hvor tid er kritisk. Sentraliserte søkemotorer vil i mange tilfeller være raskere enn rene P2P-søk i store nettverk. Mens nedlastning av informasjon vil være mer effektiv om den ikke skjer fra en tungt belastet tjener, men fra en eller flere peer-er i et nettverk. Det er viktig å ikke låse seg til èn måte å løse problemer på [26]. Desentralisering er et verktøy, og ikke et mål [7].

For MOBster sin del er det mulig at en hybrid P2P-modell slik som Napster, ville vært å foretrekke. Det er flere årsaker til dette. For det første er responstiden i MOBster dårlig, spesielt hvis antall peer-er i nettverket blir stort. Responstider på mer en 40 sekunder er uholdbart for et søk. For det andre krever forespørsler fra andre peer-er mye prosessering, noe som krever strøm og tid. Nettverkstrafikk er kostbart. Et søk mot en sentralisert terminal/PC ville kutte ned på responstider. Resultatet fra et søk kan da også sendes i sin helhet i en melding. Dette sparer tid, prosessering og penger.

Ulempene med en slik hybrid løsning er at oversikten over tilgjengelige filer på en sentralisert maskin kan være utdatert. Mobiltelefoner kan koble seg fra nettverket av forskjellige årsaker, noe som gjør at informasjonen de deler blir utilgjengelig. Et annet problem med en sentralisert løsning er at noen må drifte denne sentraliserte maskinen. En sentralisert løsning er også mer problematisk med tanke på det juridiske aspektet.

I neste avsnitt skal vi se hvordan fildeling eksisterer i dagens GSM/GPRS-nett.

7.1.2 P2P i forhold til MMS
I P2P-systemer er det ingen autoritet. Det er ingen sentralisert bruker eller maskin som styrer nettverket. I GSM-nettet er vi vant til at det er operatøren som "eier" tjenesten. Et eksempel på dette er SMS (Short Message Service). Fordelen med operatørdrevne tjenester er stor utbredelse, og en felles standard. SMS har stor utbredelse og det finnes et variert spekter med tjenester tilgjengelig.

Ulempen med operatørdrevne tjenester er at det må være en viss størrelse på kundemassen som skal ta i bruk tjenestene, for at det skal være lønnsomt for operatøren å utvikle tjenesten. Med for eksempel JXME har personer eller grupper mulighet til å lage sine egne tjenester uavhengig av operatøren. Et fotball-lag kan lage en IM-tjeneste, et korps kan lage en fildelingstjeneste der de deler ringetoner m.m.. Kostnader og administrasjon desentraliseres med P2P.

"push" i stedet for "pull"
MMS (Multimedia Messaging Service) ble introdusert i Telenor sitt nett mot slutten av 2002. MMS gjør det mulig å sende tekst, bilde, video og lyd i samme melding, fra en mobiltelefon til en annen. MMS gjør det mulig å "*pushe*" filer til andre mobiltelefoner. På den måten kan to mobiltelefoner *dele* filer med hverandre.

Den store forskjellen mellom MMS og MOBster ligger i måten brukerne tilegner seg filer fra hverandre på. MMS bruker som nevnt *push*, mens MOBster lar brukere hente (eng. "*pull*") meldinger fra andre mobiltelefoner. Forskjellen på disse to måtene å dele filer på, er ikke nødvendigvis teknologisk. MMS og MOBster representerer to helt forskjellige måter å tenke på. MMS er operatørdreven, og brukeren betaler for hver melding som sendes. MOBster er selvstendig og har ikke noen bindinger til en operatør. Brukeren betaler for hvor mye trafikk som genereres.

I neste avsnitt skal vi diskutere JXTA og tilsvarende teknologier.

7.2 Forhold mellom JXTA og liknende teknologier

JXTA er en plattform for å utvikle P2P-applikasjoner. Plattformen definerer et sett med protokoller for P2P-nettverk. Protokollene er basert på XML (Extensible Markup Language) [53]. I dette avsnittet skal vi først diskutere fordeler og ulemper med JXTA. Etter det skal vi sammenlikne JXTA med tilsvarende teknologier som .net [43], Jini [58] og Proem [14].

Fordeler og ulemper med JXTA
JXTA tilbyr et mye mer abstrakt språk enn P2P-protokoller som for eksempel Proem[34], noe som gjør det mulig å bruke en større variasjon av tjenester, terminaler og nettverkstyper i P2P-nettverk. XML er utvidbart, leselig for mennesker og kan brukes over mange forskjellige nettverk.

JXTA er fleksibelt. Det stilles ingen krav til programmeringsspråk, plattform eller nettverk. Til tross for JXTA spesifiserer de viktigste aspektene ved P2P kommunikasjon, er det ikke sikkert at JXTA egner seg like godt for alle typer P2P-applikasjoner. For en enkeltstående applikasjon kan "*overheaden*" som JXTA og XML fører med seg være mer til bry enn til hjelp, i hvert fall hvis det ikke er planer om å la applikasjonen samarbeide med andre programmer. Underspesifiseringen i JXTA gjør at utviklere må definere og kode mange selvsagte funksjoner for å få programmet til å fungere. Et eksempel på dette er JXTA sin uavhengighet av transportprotokoll. Kritikerne stiller spørsmål ved at JXTA er uavhengig av transportprotokoll, til tross for at så og si alle av dagens P2P-applikasjoner benytter TCP (Transmission Control Protocol) [4]. Dette legger mye ekstra-arbeid på utvikleren, og er ikke med på å korte ned "*time-to-market*"[35].

For utviklere handler det om å finne den rette balansegangen mellom fleksibilitet og ytelse under utvikling av en applikasjon. JXTA er kanskje ikke den mest effektive løsningen, men JXTA tilbyr en god plattform for å utvikle applikasjoner som trenger fleksibilitet til å vokse i fremtiden. [4]

[34] Proem er et sett åpne P2P-protokoller, laget med tanke på mobile *Ad-hoc* (spontane) applikasjoner og PAN (Personal Area Network, face-to-face). Se avsnitt 7.2.3.
[35] Tiden fra utviklingen av applikasjonen starter til den er på markedet, klar til salg.

JXTA og Open Source
JXTA er et *Open Source*-prosjekt, startet av Sun Microsystems. Det er JXTA Community som i dag har ansvaret for utviklingen av JXTA. Sun er sterkt inne i bildet ved å lønne ledere i prosjektet. Hva skjer hvis Sun Microsystems stopper å støtte JXTA, kan det overleve i det Community-et som eksisterer i dag [15]? JXTA sin fremtid er avhengig av de som bidrar inn i prosjektet, og av deres ferdigheter.

JXTA og IPv6
Direkte datakommunikasjon mellom to mobiltelefoner er i dag vanskelig. Årsaken til dette er at mobiloperatørene tildeler mobiltelefonene private dynamiske IP-adresser ved hjelp av NAT (Network Address Translation) i sine GPRS-nettverk. Det samme problemet eksisterer for maskiner i kablede nettverk som er plassert bak en brannmur med NAT. En måte å løse dette på er å omgå DNS (Domain Name System) og benytte en annen identifikator i tillegg til IP-adressen. Den unike peer ID som identifiserer en peer i JXTA-nettverket kan brukes til å overkomme dette problemet, og på den måten gjøre enheten "tilgjengelig" for alle. [15]

Med IP versjon 6 vil det være nok adresser tilgjengelig til å gi alle terminaler med nettverkstilknytning en IP-adresse. Beskjedne overslag sier at det vil være 1564 IP-adresser tilgjengelig for hver kvadratmeter over hele jorda [56]. Med så mange adresser tilgjengelig vil alle mobiltelefoner kunne få en fast adresse, noe som åpner for direkte kommunikasjon mellom mobiltelefoner uten nødvendigvis å bruke JXTA eller andre liknende plattformer.

7.2.1 .net
I følge Microsoft [43] er .**net** en samling Microsoft programvareteknologier for å koble sammen informasjon, mennesker, systemer og terminaler. Den tilbyr et høyt nivå av integrasjon gjennom *XML-Web services*. Under er en oversikt over de viktigste komponentene til .net:

Utviklingsplattform – rammeverkets klassebibliotek er FCL (Framework Class Library). .net er uavhengig av programmeringsspråk og tilbyr interoperabilitet mellom programmeringsspråk. Et program kan derfor bli skrevet delvis i VB (Visual Basic) og delvis i C#[36]. Koden oversettes til *Microsoft Intermediate Language* (IL) som er språk-nøytralt. [8]

Kjøretidsomgivelse - CLR (Common Language Runtime). CLR kan sammenliknes med Java VM. Største forskjell på JVM og CLR er at sistnevnte ikke interpreterer kode.

Distribuert plattform – *Web Services*. Web Services er en samling funksjoner som samles til en tjeneste og gjøres tilgjengelig på Internett for andre programmer [8]. Web Services (SOAP, WSDL og UDDI) kan også kjøres utenom .net plattformen.

JXTA's XML-baserte meldingsutveksling likner mye på Microsofts .net og SOAP (Simple Object Access Protocol). Meldingene er tekstbaserte i motsetning til teknologier som Java RMI[37], der det brukes objektserialisering for å kommunisere med andre terminaler. Meldingsutvekslingen er stort sett det eneste JXTA, Microsofts .net og SOAP har til felles. Etterhvert som flere og flere protokoller og systemer benytter XML i meldingsutvekslingen, er det klart at bare det å bruke XML til meldingsutveksling ikke nødvendigvis betyr at teknologiene har andre aspekter til felles [36].

Hensikten med JXTA og .net er grunnleggende helt forskjellig. .net fokuserer mer på den tradisjonelle klient/tjener arkitekturen for å levere tjenester. Selv om .net kan forme grunnlaget i en P2P-applikasjon, vil det føre til mye arbeid å lage en fullverdig P2P-løsning. Å utvikle en P2P applikasjon med .net ville kreve at utvikleren spesifiserte alle kjernefunksjonene i et P2P-nettverk på nytt, det vil si at utvikleren må lage funksjonene definert i JXTA på nytt. [4]

[36] C# uttales "see sharp". Dette er Microsoft sitt nye objektorienterte programmeringsspråk.
[37] Remote Method Invocation

7.2.2 Jini

Jini [58] er et sett av API-er og nettverksprotokoller som kan brukes av utviklere som skal lage distribuerte systemer organisert i føderasjoner av tjenester. Målet med Jini er å skape et spontant nettverksmiljø, hvor enheter kobler seg til og fra uten varsel og uten noen sentral administrasjon. Jini gjør det mulig for datamaskiner å finne hverandre og bruke hverandres tjenester/programvare i et nettverk uten noen slags forhåndskunnskap om hverandre. I motsetning til JXTA som er XML-basert, er Jini objektorientert.

Jini deles i to deler:

- En **infrastruktur** for å forene tjenester i et distribuert system.
- En **programmeringsmodell** for å lage pålitelige distribuerte tjenester.

Løftet om å kunne koble sammen alle typer terminaler over et hvilket som helst type nettverk er likt for både JXTA og Jini. Det er likevel flere viktige strategiske forskjeller mellom JXTA og Jini. JXTA er interoperatibel. Det vil si at JXTA er plattformuavhengig, uavhengig av programmeringsspråk, uavhengig av nettverk og uavhengig av tilbyder. Jini derimot, er en Java-sentrert teknologi. Jini bruker RMI og objekt serialisering for å kommunisere med andre terminaler. Uten støtte fra Java-plattformen (mobilitet av kode, RMI m.m.), hadde nytten av Jini vært begrenset. JXTA er uavhengig av Java og bruker XML i stedet for objekt serialisering til å utveksle data mellom peer-er i et P2P nettverk. [4] og [36]

Jini ble startet opp av Sun Microsystems. Sun ønsker å integrere og bruke Jini strategisk i fremtidige produkter, og vil derfor ha en viss kontroll over Jinis utvikling. Sun er også en stor bidragsyter i JXTA Community. JXTA er i motsetning til Jini et *Open Source*-prosjekt. Sun er bare i stand til å hjelpe til i utviklingen av JXTA. JXTA er avhengig av entusiasmen og egenskapene til JXTA Community, som utvikler teknologien.

7.2.3 Proem

Proem er et sett P2P-protokoller, laget med tanke på mobile *Ad-hoc* applikasjoner og PAN (Personal Area Network). Proem er i tillegg en utviklingsplattform med et eget kjøretidssystem og utviklingsverktøy. Utviklingsverktøyet forenkler utvikling av Proem programmer[38].

Forskerne bak Proem er veldig opptatt av de sosiale aspektene ved mobilteknologi. De mener at PAN kan være med å hjelpe oss i vår sosiale interaksjon og til og med fremme sosialt samvær med andre mennesker. Når mennesker møtes ansikt til ansikt, reduseres følelsen av stress og frykt, og følelsen av tillit og tilfredshet økes, hevder de i [14].

Proem definerer i likhet med JXTA en del konsepter. Under er en oversikt over Proems konsepter:

- **Peer**: Mobil vert som tar del i P2P-forhold.
- **Individ**: Person som eier eller bruker peer(-er).
- **Data space**: Samling av data som eies av en peer-gruppe.
- **Samfunn** (community): et sett av entiteter (peer, individ).

Entiteter blir identifisert med navn. Navn blir uttrykt med URI (Uniform Resource Identifier).

Det er mange likheter mellom Proem og JXTA. Begge bruker XML for å utveksle meldinger mellom peer-er. Både Proem og JXTA er uavhengig av hvilket nettverk de kjører over, og begge er plattformer for å utvikle P2P-applikasjoner.

Den største forskjellen mellom Proem og JXTA er at Proem kun er laget for mobile *Ad-hoc* applikasjoner og for terminaler som er i geografisk nærhet av hverandre. JXTA er laget for alt fra PC-er til mobiltelefoner, uansett hvor de måtte befinne seg i verden.

[38] Proem programmer kalles *Peerlets*

Proem er dessuten mer ”ferdig-spesifisert” og entydig enn JXTA. Et eksempel på dette er *Proem Toolkit*. *Proem Toolkit* er et utviklingsmiljø som skal hjelpe utviklere med å lage applikasjoner. Det er mange felter innen P2P som JXTA lar være å spesifisere, og som overlates til utvikler å definere for å holde spesifikasjonen enkel. Fordelen med dette er at JXTA ikke setter noen sperre for hva som er mulig eller ikke over JXTA. Ulempen er at selv utvikling av enkle applikasjoner vil kreve en god del ekstraarbeid. Proem er laget spesielt for trådløse terminaler i *Ad-hoc* nettverk, og har spesialtilpassede grensesnitt og klasser til dette formålet.

I følge [14] er det allerede laget en fildelingsapplikasjon basert på Proem. Applikasjonen ble laget for PDA-er med WLAN-tilknytning.

I neste avsnitt skal vi se nærmere på JXME, og spesielt to aspekter ved plattformen som vanskeliggjorde utviklingen av prototypen.

7.3 JXME

I rapporten har det tidligere blitt argumentert for at JXTA er en umoden teknologi. Umodenheten i JXTA påvirker JXME i stor grad. JXTA har blant annet ingen garantert levering eller ordning[39] av meldinger. JXME kjøres på begrensede terminaler, med liten mulighet til å ta hånd om garantert levering. I avsnittene under (7.3.1 og 7.3.2) diskuteres to områder (Piper og send() og create()) der JXME kommer til kort.

7.3.1 Piper

Piper eller kommunikasjonskanaler representerer virtuelle forbindelser mellom peer-er. Piper brukes til å sende meldinger mellom peer-er i JXTA-nettverket. Som nevnt tidligere i rapporten, eksisterer det to typer piper i JXTA, *unicast* og *propagate*.

Ingen garantert levering eller ordning i JXTA

Under testing av MOBster der klienter var koblet til hvert sitt relè opplevde vi at systemet oppførte seg annerledes enn forventet. Enkelte meldinger kom frem, andre ikke. Hvilke som kom frem og til hvem virket helt tilfeldig. Denne merkelige oppførselen i JXTA ble tatt opp med JXTA Community. De kunne fortelle at vilkårligheten som vi opplevde ikke var på grunn av *ProxyService* (relè), men på grunn av naturen til JXTA *propagate* (utbredelse) piper. Det tilbys ingen garanti for levering, og ingen garantert orden for hvordan beskjedene skal leveres. Tap av meldinger kan forårsakes av midlertidige sperr i nettverket, at peer-er er overbelastet eller blir utilgjengelige. Hvis pålitelighet og ordning av meldinger er påkrevet, må dette implementeres over meldingsutvekslingen, noe som krever mer ressurser. JXTA Community sammenlikner dagens implementasjon av piper med UDP (User Datagram Protocol).

MOBster bruker en *propagate* pipe til å spre søk i JXTA-nettverket. I hvilken rekkefølge slike meldinger ankommer andre peer-er har ingen betydning for søket. Garantert ordning av meldinger er derfor ikke et krav for MOBster sin del. Derimot, hvis en melding skulle bli borte vil dette ha konsekvenser for MOBster. Tap av meldinger kan føre til at søkene begrenses og at viktig informasjon blir borte. MOBster har ingen mulighet for å oppdage at meldinger blir borte.

Garantert levering av meldinger vil bli implementert i neste versjon av JXTA plattformen.

I neste avsnitt skal vi se nærmere på *send()* og *create()* fra JXME-API.

[39] Det er flere måter å ordne meldinger på. *Total ordning* vil si at meldinger ankommer *alle* mottakerene i samme rekkefølge som de ble sendt.

7.3.2 send() og create()

Som nevnt tidligere brukes *send()* til å sende meldinger/data mellom peer-er i JXTA-nettverket. *Create()* brukes til å opprette en ny enhet i JXTA. Enheten kan være av type peer, pipe eller gruppe. Send()-metoden i JXME-API tar inn fire argumenter. Disse argumentene er:

- String *name* – Navnet på pipen som meldingen skal sendes inn i.
- String *id* – PipeID til den pipen som meldingen skal sendes inn i.
- String *type* – Hvilken type pipe. Enten *unicast* eller *propagate*.
- Message *data* – data av type **Message**. En **Message** består av en tabell med **Element**-er.

Metoden returnerer også en ID som kan brukes til å finne responser på kallet, hvis det er noen.

Create()-metoden i JXME-api tar inn tre argumenter. Disse argumentene er:

- String *type* – peer, pipe eller gruppe.
- String *name* – navnet på enheten som skal opprettes
- String *arg* – avhenger av hva som skal opprettes. En pipe kan enten være *unicast* eller *propagate*.

Også denne metoden returnerer en ID som kan brukes til å finne responser, hvis det er noen.

Problemet med disse metodene er inkonsekvent bruk av argumentet *name*. Name brukes i create-metoden for å angi et navn på enheten som skal opprettes. Da vi forsøkte å benytte navnet i send-metoden hadde det ingen funksjon. Det gjorde ikke noe fra eller til om jeg brukte navnet eller ikke. Problemet ble tatt opp med JXTA Community, som kunne bekrefte det jeg hadde funnet ut.

"...once the pipe is created/discovered/resolved, the name is ignored and only the pipe id matters..."(Akhil Arora)

JXTA Community har i etterkant gått inn for å endre send()-metoden i neste versjon av JXME. Det vil derfor ikke bli mulig å angi et navn på pipen som en melding skal sendes inn i, i neste versjon.

I neste avsnitt skal vi se på prototypen som ble utviklet i forbindelse med oppgaven og diskutere de forskjellige valgene som ble tatt i forbindelse med utviklingen.

7.4 Prototypen

I løpet av prosjektet klarte jeg å realisere en P2P fildelingsapplikasjon for mobiltelefoner. Applikasjonen ble testet på mobiltelefonemulatorer. Hvor oppsiktsvekkende er en slik applikasjon? Hvor radikal er egentlig en slik innovasjon?

Innovasjonsteoriene [5] deler gjerne innovasjoner i to grupper, *inkrementelle* og *radikale* innovasjoner. En inkrementell innovasjon er en liten endring på et produkt, for eksempel en ny type låser på en bil. En radikal innovasjon er en totalt ny måte å lage et produkt på eller et helt nytt produkt. En ny lås kan isolert sett være en radikal innovasjon, men i sammenheng med en bil blir den en inkrementell innovasjon. Mange inkrementelle innovasjoner kan til sammen utgjøre en radikal innovasjon.

Sett fra et fildelings ståsted, er ikke MOBster en radikal innovasjon. P2P-fildeling har eksistert i en del år. Med MOBster blir P2P-fildeling tilgjengelig på en ny type terminal, nemlig mobiltelefonen. MOBster er derfor en radikal innovasjon sett i fra en mobiltelefon sitt ståsted. P2P-fildeling på den måten det er implementert i MOBster, har så langt vi vet, ikke eksistert tidligere på mobiltelefoner. Den eneste formen for fildeling vi kjenner til er igjennom SMS og MMS, ved at brukere kan "*pushe*" meldinger til hverandre (se avsnitt 7.1.2).

I de neste avsnittene skal vi ta for oss de forskjellige delene av prototypen, og diskutere disse i detalj.

7.4.1 Brukergrensesnitt

Mobiltelefoner har en liten skjerm, med begrensede input-muligheter. På grunn av begrenset batterikapasitet og prosessorkraft er det også begrenset hvor mye ressurser som kan brukes til brukergrensesnitt. Brukere verdsetter enkelhet [7]. Enkle, forståelige skjermbilder, som er bygget opp på en logisk måte er viktige for brukere. Brukergrensesnittet kan være det som skiller en suksess fra en fiasko.

Brukergrensesnittet har ikke stått i fokus under utviklingen av MOBster. Vi har likevel forsøkt å gjøre brukergrensesnittet enkelt og intuitivt. I en eventuell nye versjon av MOBster må det legges mye arbeid i å gjøre MOBster brukervennlig og oversiktlig.

7.4.2 Relèet

Mobiltelefonene er helt avhengige av relèet for å kommunisere med hverandre. Relèet opptrer på vegne av mobiltelefonen inn mot JXTA-nettverket (se avsnitt 5.2). Men hvem skal eie og drifte relèet? Vi ser for oss tre typer scenarier i forbindelse med eierskap til relè.

1) Egen PC

Flere og flere har tilgang på PC hjemme. Samtidig øker utbredelsen av bredbåndsforbindelser. Tall fra Statistisk Sentralbyrå (SSB) gjengitt i Dagbladet [45], viser at det finnes 1,6 millioner aktive Internettabonnement i Norge. Av disse er i underkant av 200.000 bredbåndsabonnement[40]. Personer med egen PC som "alltid" er påkoblet Internett og som har en fast IP-adresse, vil kunne ha mulighet til å bruke sin egen PC som relè. På den måten kan brukeren selv ta ansvar for vedlikehold av det utstyret han benytter. Andre mobiltelefoner vil kunne bruke brukerens maskin som relè, så lenge de kjenner IP-adressen og porten.

2) Operatør

Et annet alternativ er at mobiloperatøren leverer relè-tjenesten til sine abonnenter. MOBster vil generere mye trafikk i mobilnettet. Operatøren kan ved å tilby en slik relè-tjeneste, legge til rette for at flere tar i bruk MOBster.

Når en peer slutter å polle relèet, vil relèet ta vare på innkommende meldinger til peer-en for en viss tid. Tiden som relèet lagrer meldingene er gitt ved en "*lease*". Dette gjør det mulig å differensiere tiden innkommende meldinger blir lagret mellom peer-er. En operatør har dermed muligheten til å gi brukere som betaler for tjenesten en lenger "*lease*" enn brukere som ikke betaler. Operatøren kan bruke dette til å tjene penger på relè-tjenesten.

3) Tilfeldig

JXTA Community sier at det jobbes med å lage en tjeneste som hjelper en JXME-peer å dynamisk velge et relè. De fleste JXTA-peer-er vil kunne opptre som et relè. Problemet for en JXME-peer er å finne disse. En tjeneste som finner et relè for JXME-peer, vil kunne sikre en jevn spredning av JXME peer-er på relèene, noe som igjen kan forhindre at enkelte noder i nettet blir overbelastet.

[40] Statistisk Sentralbyrå definerer bredbånd som Internett-abonnement med overføringskapasitet høyere enn 384 kbit i sekundet.

7.4.3 Forbedringer

Det ble laget flere versjoner av prototypen før den endelige versjonen ble klar (se kapittel 6 "Prototypen"). Under utviklingen ble det tatt mange valg som har hatt mye å si for effektivitet og oppførsel. I avsnitt 7.4.3.1 til og med 7.4.3.3 er de viktigste av disse valgene diskutert.

7.4.3.1 Hvordan begrense nettverkstrafikk?

Begrense antall HTTP-kall

Det er ikke alltid det er mulig å tilfredsstille alle krav fra en bruker. For en bruker kan det være ønskelig at programmet skal være raskt og effektivt, samtidig som det skal koste lite å bruke det. HTTP-kall er tidkrevende og kostbare. Tabell 3-3 på side 36 viser en oversikt over dette. Ut fra et mål om å holde kostnadene nede, vil det være ønskelig å begrense antall HTTP-kall til det som er absolutt nødvendig. En begrensning i antall HTTP-kall vil på en annen side føre til at brukere får en tregere respons på sine søk og forespørsler i nettet. Kravene til brukerne kan dermed komme i konflikt med hverandre. Utvikleren må inngå et kompromiss mellom kostnader og effektivitet, eller la brukeren ta dette valget selv.

En metode for å begrense antallet HTTP-kall, men samtidig opprettholde gode responstider ved søk, ble innført i siste versjon av prototypen. Polle-intervallet settes "*default*" til fem sekunder. Brukeren har selv mulighet til å endre på dette. Metoden som ble innført sier at hver gang en bruker sender et søk, halveres intervallene mellom hvert poll applikasjonen foretar. I tilfellet med polle-intervall på fem sekunder senkes intervallet til to sekunder. Det nye intervallet varer i ti poll etter at søket er sendt. I løpet av ti poll bør normalt en peer ha fått noen responser på sitt søk (se Tabell 6-4 side 76). I en eventuell ny versjon av programmet bør antallet poll med halvert polle-intervall justeres dynamisk etter hvor mange svar og hvor lang tid det tok å få responser forrige gang brukeren gjennomførte et søk. I Eksempel 7-1 er et forslag til hvordan en slik algoritme kan se ut:

Algoritme for beregning av antall poll

1. Polle-intervall settes *default* til fem sekunder. Antallet poll settes *default* til 10.
2. Når et søk foretas settes det i gang en stoppeklokke og en teller. Stoppeklokken teller antallet sekunder til en respons mottas. Telleren teller antall responser.
3. Når programmet mottar en melding, sjekker den om den inneholder en respons. Gjør den det inkrementeres telleren, og stoppeklokken fortsetter. Hvis meldingen er tom, stoppes klokken og telleren.
4. Hvis de to neste påfølgende meldingene er tomme, anses søket som avsluttet. Hvis det kommer en ny respons startes klokken og telleren. Tiden på klokken økes med henholdsvis ett eller to polle-intervall.
5. Det nye antallet poll som mobiltelefonen skal polle med halvert polle-intervall er gitt ved formelen:

$$n_{p(i)} = (2 \times n_{p(i-1)} + n_r) / 3$$

Der n_p er antallet beregnet poll med halvert polle-intervall og n_r er antallet registrerte responser ved forrige søk. Forrige beregnede antall poll vektlegges mer enn registrert verdi.

Eksempel 7-1 Algoritme for beregning av antall poll

Metoden for å begrense antall HTTP-kall vil kunne være med på å redusere nettverkstrafikken og også kostnadene ved bruk av applikasjonen. Metoden vil kunne føre til en noe tregere respons, men dette vil kunne begrenses ved å bruke dynamiske polleintervall og algoritmen beskrevet i Eksempel 7-1.

Den endelige løsningen på problemet vil kunne komme med MIDP 2.0, hvis JXME velger å integrere denne i plattformen. Polling mot relè blir historie når relèet kan *pushe* meldinger til mobiltelefonen.

Droppe meldingene "ingen fil er funnet"
I de første versjonene av prototypen ble det alltid returnert en respons til den som initierte søket uansett om filen ble funnet eller ikke. Dette var misbruk av ressurser. Hvis mange peer-er ikke har den etterspurte filen, fører det til unødvendig bruk av nettverket. Meldingene om at "ingen fil er funnet" bruker ikke bare opp ressurser, de er også med på å forsinke de responsene som brukeren har nytte av. I siste versjon av prototypen, valgte jeg derfor å fjerne denne funksjonen. En melding om at en fil ikke er funnet, er lite nyttig for brukeren.

Hvor lenge skal applikasjonen vente på en respons?
Hvor lang tid bør en bruker vente på responser på søket? Slik prototypen er i dag kan brukeren risikere å vente til evig tid. Det er ingen begrensninger på når et resultat fra et søk kan ankomme mobiltelefonen. I en eventuell ny versjon av prototypen vil dette måtte begrenses enten med en stoppeklokke, en teller eller en kombinasjon av disse.

7.4.3.2 Meldingsstørrelser

I avsnittet over snakket vi om hvor viktig det var å begrense antallet HTTP-kall. HTTP-kall tar lang tid og er kostbare i den forstand at det genererer trafikk. Like viktig som det er å tenke på antallet HTTP-kall som foretas, er det å tenke på dataene som overføres i forbindelse med kallene. Størrelsen på meldingene som sendes har mye å si for hvor mye datatrafikk som genereres. Hvordan kan meldinger sendes mest mulig effektivt mellom MOBster-peer-er? Hvordan kan størrelsen på meldingene som sendes begrenses?

I et nettverk bestående av hundrevis av peer-er vil størrelsen på meldingene som sendes mellom dem ha mye å si for trafikkbelastningen i nettverket. Hvordan kan kommunikasjonen mellom peer-er skje mest mulig effektivt? Under er det listet opp en del muligheter, som så blir vurdert ut i fra følgende kriterier:
A) Antallet meldinger skal være lavest mulig for ikke å belaste nettverket unødig eller påføre brukerne ekstra kostnader. Færre meldinger fører også til mindre prosessering på mobiltelefonene og mindre prosessering sparer strøm.
B) Meldingene som sendes skal være kortest mulig. Større meldinger krever mer av nettverket i tillegg til at de er dyrere å overføre.
C) Systemet må være skalerbart. Et system er skalerbart hvis det forblir effektivt etter en signifikant økning i antall brukere og ressurser.

For å kunne få til kommunikasjon mellom peer-er må peer-ene være i stand til å svare på hverandres meldinger. Fordi pipeID til de forskjellige peer-ene ikke ligger vedlagt i meldingene som sendes, er dette problematisk. Årsaken til at JXTA Community har unnlatt å gjøre pipeID-en til avsender tilgjengelig i meldingene, er vi ikke sikre på. En av grunnene kan være at JXTA Community ønsker å holde meldingsstørrelsen så liten som mulig, og heller la utviklerne implementere dette selv hvis de har behov for det. Under er det skissert fire mulige løsninger på problemet:

1) Returnere svaret i samme pipe som søket ble mottatt i (MOBSTER_PIPEID)
Et innkommende søk inneholder navnet på avsenderen og URI-en (unik) til pipen som meldingen ble sendt inn i. Navnet på avsenderen kan velges av brukeren selv, og er ikke en unik identifikator. Responser sendes tilbake til avsender igjennom samme pipe som søket ble mottatt. Meldingen merkes med navnet på den som sendte søket. Avsenderen vet at meldingen er til seg ved å sjekke om navnet matcher og responsen tilsvarer søket som han selv sendte av gårde.

Det er mange ulemper ved å gjøre det på denne måten. Hver eneste peer (kan være mange tusen) som lytter på MOBster-pipen vil motta responsen, fordi mobiltelefonen ikke får sjekket om meldingen er til seg før den har sett innholdet i meldingen. Dette vil føre til unødvendig mye trafikk i nettet, og misbruk av ressurser. I verste fall kan peer-er stå og motta mange tusen meldinger som ikke har noen relevans for dem i det hele tatt. Dette koster penger og tid. Alle peer-er må da prosessere hver melding som hver enkelt peer sender ut. Selv med bare 10-15 peer-er i nettverket er denne løsningen uakseptabel. Løsningen er ikke skalerbar, og oppfyller verken A eller C.

2) Innhente pipe_ID til hver eneste peer som lytter til MOBSTER_PIPE_ID og beholde informasjonen i en "*buddylist*" på hver enkelt peer
I dette tilfellet kaller hver enkelt peer *search*()-metoden i JXME-API. Den gjør det mulig å søke etter alle peer-er som hører til en gruppe eller har samme navn[41]. URI-en til de forskjellige peer-ene vil bli returnert til mobiltelefonen som kalte *search*(). De forskjellige URI-ene lagres i en liste. Listen brukes som et oppslagsverk hvis peer-en senere har behov for å sende en respons til noen i listen.

Fordelen med denne fremgangsmåten er at respons på søk kun ankommer den som sendte søket og ingen andre. Ulempen er at den krever mye minne på mobiltelefonen. Mobiltelefoner har begrenset minne og plass til MIDlet-er (64kB for Nokia 6610). En URI er 128 bit lang, og med tusenvis av peer-er i et nettverk vil *buddylisten* bli enormt stor. Den vil ta opp all kapasitet på mobiltelefonen. Et annet problem er at alle meldingene som sendes i forbindelse med innhenting av URI i oppstartsfasen vil oversvømme nettverket. Hver ny peer som kobler seg til nettverket skal motta URI-en til alle de andre som er koblet til. En siste ulempe er at systemet blir veldig lite fleksibelt. Om peer-er kobler seg av eller forsvinner, vil ikke *buddylisten* klare å få med seg det. Etter kun kort tid vil informasjonen i *buddylisten* være ubrukelig. I mindre nettverk med kun få peer-er vil løsningen kunne fungere. Det er denne løsningen som brukes i Chat-eksempelet til JXTA Community. Løsningen er lite skalerbar og lite fleksibel og tilfredsstiller derfor ikke kriteriene i A og C.

3) I hver eneste melding som sendes hektes det på et *Element* som inneholder pipeID-en til avsenderen
I dette tilfellet utvides alle meldinger som sendes med ett *Element*. I elementet lagres den unike pipeID-en til avsenderen. Mottaker kan enkelt sende responser til avsender på pipen som er opprettet.

Ulempen med denne løsningen er at hver eneste melding som sendes over nettverket blir lenger enn nødvendig. For små meldinger som kun inneholder korte kommandoer kan overhead bli så mye som 30-40%. For lengre meldinger (> 1000 Byte) vil ikke dette være merkbart. Fordelen med denne løsningen er fleksibiliteten og skalerbarheten. Systemet fungerer fint uansett hvor stort nettverket blir. Hver enkelt melding som beveger seg i nettverket inneholder all informasjon som behøves. Denne løsningen ble benyttet i MOBster, til tross for at den ikke oppfyller kriteriene i B.

4) Lage en navnetjeneste tilknyttet relèene
Det er flere måter å tilknytte en navnetjeneste til relèet på. En måte er å lage en *servlet* som arbeider på samme maskin som relèet. En annen måte kan være å integrere hele navnetjenesten i relèet, og da også i JXTA.

En navnetjeneste skal holde oversikt over alle peer-er som er koblet til nettverket, og er egentlig en sentralisert utgave av *buddylisten* fra punkt 2. Det er lettere å holde en sentralisert liste oppdatert enn mange lister som er spredt rundt omkring på de forskjellige peer-ene. Et relè har dessuten mye større minne og bedre forutsetninger for å kunne lagre en slik liste. Peer-er som er koblet til relèet bruker navnetjenesten når den skal sende meldinger.

[41] Et eksempel på dette er at alle peer-ene får et navn på formen `mobster.<brukernavn>`.

Problemet med denne løsningen er at den ikke er fleksibel. I tillegg er den med på å bevege MOBster lenger vekk fra en ren P2P-løsning og mot en mer sentralisert løsning. Systemet får også økt kompleksitet. Løsningen tilfredsstiller ikke kravet i C.

7.4.3.3 Øvrige endringer

Det ble også gjort en del mindre viktige endringer på prototypen. Disse endringene er listet opp under:

- mindre sammensetting av String-er ("*String-concatenation*"). Bruk av '+' operator til å sette sammen objekter fører til unødvendig bruk av minne og prosessorkraft.
- små endringer av brukergrensesnittet for å gjøre programmet mer oversiktlig og enklere å bruke.
- fjernet kommentarer i koden, fordi dette tar opp plass og ikke har noe å si for programmets virkemåte
- gjennomført tipset om å aldri initialisere et objekt til null.
- lagre filer sortert etter navn (alfabetisk), fordi dette gir en bedre oversikt over filene som er lagret på telefonen.

I avsnitt 7.4.3 har vi sett på forbedringer som ble gjort med prototypen. I neste avsnitt skal vi se på skalerbarheten til systemet slik det er i dag og komme med forslag til hvordan den kan forbedres.

7.4.4 Skalerbarhet

Et system er skalerbart hvis det forblir effektivt etter en signifikant økning i antall brukere og ressurser. Slik MOBster er i dag sendes søk ut til alle peer-er som lytter på MOBster-pipen. Et slikt søk gir mange responser, og er på den måten med på å bedre kvaliteten på søket. Kvaliteten på et søk bestemmes av hvor raskt søket er, hvor treffsikkert det er og hvor mange prosent av peer-ene som deltar i nettverket det søkes hos. Denne måten å søke på kan imidlertid være problematisk når nettverket blir stort. Antall meldinger m som går i nettverket som følge av et søk er gitt ved formelen:

$$m = 2 \times (n-1)$$

der n er antall peer-er i nettverket. I et lite nettverk med fire peer-er blir det seks meldinger for hvert søk. Vokser nettverket til 1000 peer-er, vil hvert eneste søk som sendes generere 2 x (1000-1) = 1998 meldinger. Hvis alle peer-er foretar et søk samtidig, vil det gå nærmere to millioner meldinger i nettverket. Det sier seg selv at dette ikke er noen holdbar løsning for store nettverk. Systemet er lite skalerbart slik det er i dag. Metoder for å begrense søkene må innføres. Vi vil se på ulike forslag til slike metoder. Først en kort repetisjon av hvordan søk fungerer i MOBster.

Som nevnt tidligere må alle JXME-peer-er polle JXTA-relèet jevnlig for å se etter innkommende meldinger. Det er bare mulig å hente ned en melding av gangen med dagens versjon av JXME. Brukeren kan selv angi hvor ofte han ønsker å polle relèet. Intervallet kan være alt fra ett sekund og oppover. I nettverk med få peer-er vil ikke disse begrensningene by på særlig store problemer. Det er alltid en viss forsinkelse forbundet med å sende ut meldinger (et par sekunder). Etter at søket er sendt, må de forskjellige mottakerene laste ned forespørselen fra relèet (se Figur 6-10). Det tar gjerne fire til fem sekunder før mottakerene har mottatt søket. Etter det står det meste og faller på peer-en som sendte forespørselen. Peer-en må da stå å polle helt til alle svarene er kommet tilbake. Med et polle-intervall på ett sekund tar det minst 1000 sekunder å avslutte søket i ett nettverk med 1000 peer-er.

Under er det listet opp tre forslag til hvordan søk kan begrenses.

1) Vente en maksimal tid på svar

Etter å ha ventet en gitt tid, slutter peer-en å lytte på pipen sin. Meldinger som ankommer relèet etter dette kommer ikke frem til peer-en. Peer-en kan jobbe videre med andre oppgaver.

Dette forslaget er en god ide for den enkelte peer. Peer-en blir spart for mye prosessering. Forslaget er derimot ingen god løsning for systemet som helhet. Alle andre peer-er i nettet vet ikke når denne tidsfristen går ut og må dermed prosessere meldinger som likevel blir kastet. Dette fører til unødvendig nettverkstrafikk og sløsing med prosessortid, batterikapasitet og penger.

2) Innføre et Time-to-Live (TTL) felt i meldingene som sendes
Som nevnt i avsnitt 2.2.4 har Gnutella en funksjon for å avgrense søk. Ved hjelp av et TTL (Time-To-Live)-felt, begrenses antall noder som mottar søket [13]. TTL-feltet kan også være med på å hindre at meldinger blir svevende rundt i nettet til evig tid og at meldingene belaster nettet unødig. TTL-feltet minker med èn for hver node den passerer. Verdien i TTL-feltet er i utgangspunktet satt til syv (se avsnitt 2.2.4). Gnutella omtaler dette som en *horisont*. Det er flere måter å lage en horisont på. Et TTL-felt er en mulighet, men vil ikke ha særlig stor nytte i JXME. Årsaken er at mobiltelefonene som utfører søket er "kant"-peer-er, som ikke ruter meldingen videre. En annen mulighet er å avgrense søket til alle JXME-peer-er tilknyttet samme relè. Også dette blir vanskelig, fordi det ikke finnes noen funksjoner i JXME for å finne ut nettopp dette.

En annen måte å bruke TTL-feltet på er å benytte det som et slags tidsstempel. Når meldingen sendes, settes det et tidsstempel for når meldingen ikke lenger er gyldig. Alle noder som behandler meldingen undersøker om tidsstempelet er gått ut. Hvis tidsstempelet er gått ut, forkastes meldingen.

Problemet med en slik løsning er synkroniseringen av klokker i nettverket. JXTA er et distribuert system. Klokker på de forskjellige terminalene som deltar i systemet kan variere og drive i forhold til hverandre. Dette kan føre til at meldinger som forkastes av en peer kan godkjennes av en annen, fordi peer-ene har forskjellig oppfatning av hva tiden er. Det er ikke mulig at alle parter i et slikt system kan bli enige om hva klokka er på et gitt tidspunkt [16]. Det er mulig å løse dette med NTP (Network Time Protocol) eller logiske klokker og "hendte-før" relasjoner. Dette er utenfor fokuset til denne oppgaven, og kan leses mer om i [16].

3) Kunne opprette grupper og søke innad i gruppene.
En tredje løsning er å la klientene organisere seg i grupper. Søkene foretas kun innad i en gruppe. Det settes som krav at en bruker er med i en gruppe for å kunne søke etter filer og dele ut sine egne. JXME har støtte for å opprette, bli med i og forlate grupper. Grupper kan være en god måte å begrense søk på. Det finnes flere muligheter for å organisere slike grupper. En mulighet er at alle brukere selv står fritt til å opprette sine egne grupper. En vennegjeng eller en skoleklasse kan ta initiativ til å lage en slik gruppe. Brukerne kan dele filer innad i gruppen i de sammenhengene de befinner seg i. Dette er med på å begrense antallet terminaler som deltar i behandlingen av søk og dermed også antallet meldinger i nettverket.

En annen mulighet er at en operatør, ISP eller liknende oppretter en rekke grupper på forhånd. Gruppene kan ha forskjellige temaer. Noen grupper kan ha fokus på video, noen på klassisk musikk og andre på pop-musikk. Brukere kan selv velge hvilke grupper de ønsker å delta i. Grupper vil generelt kreve en del ekstra administrasjon. En slik løsning vil kunne by på juridiske problemer (se avsnitt 7.7 "Juridiske aspekter").

En tredje mulighet er at grupper opprettes på grunnlag av fysisk nærhet. Det er ingen støtte for dette i JXME. BlueTooth kunne vært en egnet teknologi til et slikt scenario. Med BlueTooth kan brukere oppdage og kommunisere med andre BlueTooth-terminaler innenfor en radius på 10 meter. Nettet settes opp kun for anledningen i et såkalt *Ad-hoc*-nett. Fordelen med et slikt nettverk er at det blir begrenset både geografisk og med tanke på antall terminaler som deltar. Ulempen er at brukerne av nettet er mobile, og fort kan bevege seg utenfor nettverkets *horisont*. En slik løsning kan begrense mobiliteten til brukerne og nytten av nettverket. Tilgangen på informasjon i nettverket blir variabel.

Dette er tre mulige måter å begrense søk på. Metoder for å avgrense søk er forslag til videre arbeid. I neste avsnitt skal vi se på hvilke kostnader datatrafikken påfører brukeren.

7.4.5 Kostnader

Gjennom hele kapittelet har det vært fokus på begrensninger og kostnader forbundet med MOBster. Det har vært snakk om å begrense antall HTTP-kall (7.4.3.1), begrense meldingsstørrelser og begrense rekkevidden av søk. Tabell 6-2 og Tabell 6-3 på side 76 viser en oversikt over hvor mye trafikk som faktisk sendes og mottas i løpet av 15 minutters normal bruk av applikasjonen. Tabell 7-1 viser en oversikt over hvor mye denne trafikken vil koste ut i fra prisene i Tabell 3-1 og Tabell 3-2 på side 35.

	Polling hvert 2.sekund	Polling hvert 5.sekund
GSM-data (Telenor) *	kr. 13,35	kr. 13,35
HSCSD (Telenor) *	kr. 28,20	kr. 28,20
GSM-data (Netcom)	kr. 13,35	kr. 13,35
HSCSD (Netcom) **	kr. 26,70	kr. 26,70
GPRS (Telenor) ***	kr. 22,48	kr. 11,18
GPRS (Netcom storbruker) ****	Kr. 3,37	kr. 1,67

* Ikke medregnet etableringsgebyr på 100,- og månedsavgift på 15,-
** Antatt bruk av to kanaler
*** Uten etableringsgebyr. Antar at dette er av de første 0,5 MB som overføres.
**** Første 20MB koster 200,-. Bruker priser for datatrafikk som overgår 20MB.

Tabell 7-1 Priser på trafikk i MOBster, basert på bruk i 15 minutter

I tilfellet med polling hvert femte sekund utgjør tomgangstrafikken omtrent 60% av totaltrafikken. Med polling hvert andre sekund utgjør tomgangstrafikken over 80%. Filene som ble brukt under testingen var på 200 byte hver. Med større filer vil også kostnadene øke.

Økonomi er ikke fokusområdet til denne oppgaven. Når pris og kostnader får betydning for de teknologiske valgene, er det likevel verdt å ta i betraktning. Det koster å delta i et nettverk. Selv om du bare laster ned tre små filer i løpet av et kvarter må du betale mellom 13 og 26 kroner for dette. Brukeren må dessuten selv betale for at andre søker på brukerens mobiltelefon. Det gjenstår å se om brukere vil godta dette. Vi regner med at prisen på datatrafikk i mobilnettet vil falle i pris etterhvert som neste generasjons mobiltelefonsystem (3G) introduseres.

Pris på datatrafikk vil ha mye å si for bruken av en fildelingsapplikasjon på mobiltelefoner. Blir det for dyrt vil muligens brukerne heller laste ned musikk hjemme eller på skolen, og overføre filene til mobiltelefonen selv.

I neste avsnitt skal vi se på alternative måter å realisere en P2P fildelingsapplikasjon på mobiltelefoner på.

7.4.6 Alternative løsninger

JXME er ikke den eneste plattformen utviklere kan bruke for å realisere fildeling. Tidligere i dette kapittelet snakket vi om hvordan MMS kunne brukes til å dele filer. MMS kan bare tilby fildeling ved at brukere ”*push*-er” meldinger til hverandre. I avsnittene under skal vi se på plattformer for å utvikle P2P-fildeling med ”*pull*”, det vil si analogt til MOBster.

Symbian/C++
Symbian OS (Operating System) er et operativsystem utviklet spesielt med tanke på mobiltelefoner. Symbian er eid av de største mobiltelefonleverandørene: Ericsson, Panasonic, Motorola, Nokia, Psion, Samsung, Siemens og Sony Ericsson.

Symbian OS er skrevet i C++ [63]. C++ er derfor det foretrukne programmeringsspråket på Symbian. Med C++ kan utviklere få tilgang til store deler av Symbian OS API. Versjonen av C++ som brukes på mobiltelefoner er noe begrenset i forhold til versjoner for PC-er. For de fleste applikasjoner har C++ god nok ytelse. Skulle det være spesielle oppgaver som ikke går raskt nok i C++, har Symbian støtte for C og *Assembler*.

Fordelen med å bruke C++ på Symbian er at utviklere får direkte tilgang til OS-et's API. Til tross for at JXTA er språkuavhengig og det allerede finnes en C++ versjon av JXTA, vil ikke en fullversjon av JXTA i C++ kunne kjøre på en mobiltelefon, på grunn av mobiltelefonens begrensninger. Ulempen med å benytte Symbian/C++ i forhold til JXME, er at all P2P-funksjonalitet som finnes i JXTA, må implementeres på nytt. En utvikler måtte ha laget hele systemet fra bunnen av.

JXTA Community jobber med å lage en C-versjon av JXME. En C-versjon av JXME vil fungere omtrent likt som Java-versjonen. All kommunikasjon med relèet vil foregå over HTTP.

MIDP 2.0 / "Push registry"
I avsnitt 3.6 tok vi for oss MIDP 2.0. Vi så på de nye mulighetene versjon 2.0 kommer til å tilby: økt sikkerhet, bedre grafikk og bedre nettverksmuligheter. Med MIDP 2.0 er det mulig å "*pushe*" meldinger ut til en mobiltelefon. Det vil ikke være behov for polling. Pollingen og tomgangstrafikken står for en betydelig del av den totale trafikken. P2P-fildeling vil kunne realiseres ved at peer-ene "*pusher*" søk og forespørsler via relèet til deltakere i nettverket. Fildelingen vil skje på samme måte som i MOBster.

Dagens versjon av JXME bruker MIDP 1.0. JXTA Community har ennå ikke tatt stilling til når de kommer til å gå over til den nye versjonen av MIDP, men dette skjer trolig i løpet av sommeren. Neste versjon av JXME kommer til å bli hetende JXME 2.0.

JXME 2.0
Lederen for JXME-prosjektet i JXTA Community, Akhil Arora, har nylig tatt initiativ til å sette opp en plan for JXME 2.0. Versjon 2.0 av JXME har som mål å endre plattformen slik at den er kompatibel med versjon 2.0 av JXTA. JXTA 2.0 ble sluppet i begynnelsen av 2003. Slik det er i dag kan ikke en JXME-peer kommunisere med en JXTA 2.0 peer.

JXME-prosjektet skal innen slutten av mai lage en ny versjon av *HttpMessenger* for CLDC og CDC, som gjør at JXME-peer-er kan snakke ved hjelp av JXTA 2.0 relè protokoll. De har også som mål å få laget en TCP Messenger som tillater mobiltelefoner med støtte for TCP å koble seg til relèet ved hjelp av TCP. Dette vil føre til en mer effektiv utnyttelse av nettverket, se [59].

Akhil understreker også viktigheten av å teste implementasjonen nøye på mobiltelefoner. Hver mobiltelefon implementerer støtte for HTTP på litt forskjellige måter. Dette gjør det vanskelig å få den samme koden til å fungere likt på alle typer telefoner. Dette var noe vi fikk erfare under vår testing av MOBster på Nokia mobiltelefoner (se 6.5.2).

7.5 Mobilitet

Denne rapporten har hatt et sterkt fokus på begrensningene som ligger i mobiltelefoner og J2ME. Når det gjelder mobiltelefoner og applikasjonsutvikling er det lett å fokusere på begrensningene. Det er lett å glemme alle fordelene mobiltelefoner har. I avsnittene under skal vi diskutere mobilitet i forhold til MOBster.

7.5.1 Flytte applikasjoner fra stasjonære til mobile omgivelser

Nye teknologier og nye måter å gjøre ting på krever alltid en viss tilvenningsperiode. Det tok tid for kontorpersonell å bevege seg vekk fra skrivemaskin og over til PC. PC-en var en helt ny måte å tenke på. Det krever mye ressurser å introdusere et nytt produkt på markedet. Store kampanjer på TV, radio, i aviser og på Internett er ofte nødvendig. Det kan ta lang tid før et produkt er inne på markedet, hvis det i det hele tatt kommer så langt.

Per dags dato eksisterer det få tjenester for mobiltelefoner utenom tale og SMS. Det meste dreier seg om logoer, ringetoner og bildebeskjeder. Ved å ta en tjeneste som eksisterer på stasjonære maskiner, og som folk er godt vant til å bruke, og flytte over i en mobil kontekst, vil produsentene kunne spare mye krefter på markedsføring. Ta for eksempel MOBster. Folk har etter hvert blitt vant til å bruke fildelingsapplikasjoner på PC-er. Mange har programmer som KaZaA eller liknende installert på sin hjemme-PC. Brukerne vet hva de må gjøre for å søke etter en fil, laste denne ned og bruke den. I dette prosjektet har vi overført en tjeneste som normalt bare finnes på PC-er til en mobiltelefon. Den har de samme funksjonene som på en PC, bare i en litt annen innpakning. Vi påstår at et slikt program lettere vil bli tatt i bruk enn andre programmer som brukerne ikke kjenner fra før. ”Alle” vet hva fildeling er. P. Luff og C. Heath [29] understreker også viktigheten av familiaritet med mediet (mobiltelefonen) som brukes.

Det er heller ingen ulempe at det finnes så mange mobiltelefoner som det gjør. Det gjør en eventuell diffusjonsprosess enda raskere.

7.5.2 ”Place” og ”space”

I litteraturen om mobilitet har det vært mye snakk om forskjellene mellom ”*place*” og ”*space*”[42]. Et ”space” er et rom, en bygning eller sted. Med noen rom er det forbundet handlinger. Handlingene som er forbundet med rommet gjør et ”space” til en ”place”. Et eksempel på dette er soverommet. Soverommet er et soverom fordi vi sover der. [27]

Hvis noen ringer til deg på hjemmetelefonen, starter som regel samtalen med ”Hvordan har du det?” eller ”Hvordan går det?”. Hvis noen ringer deg på mobiltelefonen starter samtalen ofte med: ”Hvor er du?” og ikke ”Hvordan har du det?”. Når vi ringer hjem til noen og de tar telefonen, regner vi med at de er hjemme (de kan ha viderekoblet telefonen). Vi vet hvilken kontekst de er i. Det forventes en viss oppførsel i et visst miljø. Mobile terminaler har begynt å fjerne grensene mellom ”space” og ”place”. Det er mulig å ta med seg arbeidet over alt, til og med soverommet kan bli en arbeidsplass.

“Everywhere becomes everyplace all the time” P. Agre [27].

Med MOBster er det mulig å laste ned filer på steder der det tidligere ikke var mulig. Brukeren tar med seg fildelingen der hvor han er. Dette åpner for nye måter å tenke fildeling på. Ta for eksempel tillit. Med scenario fire beskrevet i avsnitt 6.1.4 har brukere mulighet til å se den de laster ned filer fra. Det vil kunne gi en helt annen følelse av trygghet og tillit [14]. Med KaZaA og liknende programmer vet ikke brukere hvem de laster ned i fra. De som deltar er anonyme, og det er ikke mulig å vite om dette er personer med uhederlige hensikter. Med fildeling ”ansikt-til-ansikt” vil brukerne kunne oppleve en helt annen tillit til systemet.

[42] ”Place” og ”space” er vanskelig å oversette til norsk, vi velger derfor å bruke de engelske ordene og forklare disse i detalj

7.5.3 Sosialt

Vil MOBster endre brukernes oppførsel? Rheingold [35] snakker om hvor viktig det sosiale og mellom-menneskelige er. Det sosiale må stå i sentrum for utvikling av ny teknologi. Teknologi er sekundært. Når brukere kan dele filer på T-banen, vil dette endre deres oppførsel? Med tanke på at mobiliteten til brukeren økes, vil dette kunne endre brukerens oppførsel. Det er viktig å designe *for* mobilitet, og ikke *mot* mobilitet [41]. En bruker som deler filer med en mobiltelefon vil antakeligvis treffe flere mennesker (fysisk) enn brukeren som sitter hjemme. En fildelingsapplikasjon på mobiltelefonen vil være med på å ta vare på mobiliteten som brukeren hadde før han tok applikasjonen i bruk. Brukeren blir ikke bundet til et sted. Mobiliteten til brukeren kan for eksempel gjøre at han treffer personer han ellers ikke ville truffet. Mobiliteten fører til mellom-menneskelig kontakt.

Det er viktig at utviklere tar høyde for at brukere kan bruke systemet på andre måter enn det er tiltenkt [20]. Eksempler på dette er at personer kan bruke programmet til å spre virus eller "overvåke" andre brukere. Hvis en P2P-fildelingstjeneste bruker lokasjon til å begrense størrelse på nettverk, vil en bruker som har kjennskap til en annen person sitt brukernavn, kunne "overvåke" den andre personen, ved å se om de befinner seg i nærheten av hverandre.

I neste avsnitt skal vi se på en fildelingsapplikasjon for mobiltelefoner, som er utviklet av et fransk selskap som heter Apeera.

7.6 Relatert arbeide - Apeera

I september 2002 skapte et nytt produkt store overskrifter i aviser rundt omkring i Europa. Et fransk selskap som heter Apeera [49] hadde utviklet en teknologi som skulle gjøre det mulig å dele filer i et mobilnettverk. Siden den gang har det ikke vært så mye å høre om Apeera.

Mobiltelefoner har begrenset minne. Apeera ønsker å gi brukerne av tjenesten tilnærmet ubegrenset tilgang til lagringsplass. Tjenesten kan brukes av alle som har WAP (Wireless Application Protocol) på mobiltelefonen.

På hjemmesiden til Apeera kan vi lese at klienten skal ha støtte for de fleste plattformer og operativsystemer (Symbian, PocketPC, J2ME, Palm). Med Apeeras løsning lagres filer og applikasjoner på et personlig oppbevaringssted (*repository*) på en sentral tjener, administrert av operatøren (for eksempel Telenor Mobil eller NetCom). Brukeren kan administrere sine egne filer enten gjennom et grensesnitt på telefonen eller gjennom en portal på Internett. Ved å fjerne behovet for å lagre filene på håndsettet, kan operatører tilby "uendelig" lagringskapasitet som er personlig for hver enkelt bruker. Dette er et produkt for nettverksoperatører. Det taler for at operatøren skal ha full kontroll med det som skjer.

Filosofien bak Apeera er grunnleggende forskjellig fra MOBster. Med Apeera er det i virkeligheten en PC eller tjeneren til nettverksoperatøren som tar hånd om fildelingen. Alle filer som brukeren laster ned, lastes ned til denne sentraliserte tjeneren. Filene som brukeren deler med andre mobiltelefoner, ligger lagret på tjeneren. Mobiltelefonen fungerer som en fjernkontroll av brukerens personlige oppbevaringssted på tjeneren. Apeera likner veldig på det første scenariet som ble beskrevet i avsnitt 6.1. Med MOBster skjer fildelingen direkte på mobiltelefonen. Filene lagres direkte på telefonen.

I neste avsnitt skal vi se på det juridiske aspektet ved P2P fildeling.

7.7 Juridiske aspekter

Det har vært mye oppmerksomhet rundt P2P fildelingsapplikasjoner de siste årene. Rettssakene mot Napster er eksempel på dette. Dessverre blir P2P-fildeling ofte assosiert med ulovlig distribusjon av musikk, video og programvare.

P2P-systemer har en stor fordel i forhold til sentraliserte klient/tjener systemer. Ingen kan hindre meg som enkeltperson å dele filer som ikke er opphavbeskyttet eller som jeg har produsert selv. Hvis jeg velger å dele andre filer som er opphavbeskyttet, må jeg ta konsekvensene av det. Med et delvis sentralisert system som Apeera (se avsnitt 7.6), hvor brukere kan laste ned filer til en tjener som driftes av en nettverksoperatør, vil operatørene få store problemer med ulovlig materiale som ligger lagret på deres maskin.

En ulempe med P2P systemer er at peer-er kan føle det ukomfortabelt å ikke vite hva slags forespørsler og data som sendes igjennom deres system [17]. Noder i nettverket kan uten å vite det være med på å videreformidle ulovlige data. Mobiltelefoner er kant-peer-er, og er derfor ikke med på å videresende meldinger for andre peer-er.

Enkelte frykter også at anonymiteten i P2P nettverk skal kunne resultere i misbruk. Eksempler som nevnes i [24] er pornografi og terrorisme. Terrorister kan distribuere og dele lovstridig materiale enklere i et anonymt nettverk.

7.8 Fremtidsscenarier

I avsnittene 7.8.1 til 7.8.3 tar vi for oss en del scenarier som vi mener er interessante med tanke på fildeling på mobiltelefoner i fremtiden. Scenariene har ikke direkte noe å gjøre med Peer-to-Peer fildeling.

7.8.1 Lagre filer eksternt

En av de mange begrensningene i dagens mobiltelefoner er lagringskapasitet. I MOBster er det kun 30kB tilgjengelig for lagring av eksterne data. Dette gjør at nytten av programmet blir svært begrenset, uansett om musikkfiler blir tilgjengelige eller ikke.

Det finnes løsninger på dette. Mange av dagens telefoner har BlueTooth innebygget. BlueTooth tilbyr overføringskapasitet på rundt 1 Mbit/s for terminaler som er innen 10 m radius. Det finnes bærbare harddisker på flere GB, som kommuniserer over BlueTooth. Hvis brukeren har en slik i lomma, vil lagringsproblemet være løst i lang tid fremover.

Sony har akkurat sluppet en 1 GB *memorystick* [65]. Hvis det etterhvert skulle bli mulig å bruke slike minnebrikker i telefoner, vil nok 1 GB kunne løse lagringsproblemet for en tid fremover.

7.8.2 Buffring i nettet

Søking krever mye ressurser i nettet, både av den som sender søket og av de som behandler det. En metode for å begrense ressursbruken kan være å bruke en slags form for buffring i nettet. Buffringen kan skje på relèet. Hvis relèet mottar et søk som det har vært borte i tidligere, returnerer det svaret direkte uten å spre det i JXTA-nettverket. En ulempe med en slik løsning er at svarene som ligger lagret etter hvert blir utdaterte. Det må lages funksjoner for å undersøke at informasjonen som ligger i bufferet er oppdatert. Dette likner litt på oppgavene som en *web-proxy* har i et nettverk. Nettsider som ofte blir etterspurt, blir buffret i proxyen for å begrense nettverkstrafikken ut av bedriften eller organisasjonen.

7.8.3 Mobil agent

En mobil agent er et kjørende program (både kode og data) som beveger seg mellom maskiner på et nettverk, mens den utfører operasjoner på vegne av en maskin. Et eksempel kan være å samle sammen informasjon, som til slutt returneres til den som eier agenten. En mobil agent kunne i MOBster blitt brukt til å utføre søk på vegne av peer-er. Agenten kunne beveget seg fra peer til peer og undersøkt om filene som avsender etterspør finnes. Til slutt kunne agenten returnert med resultatene til avsender. Avsenderen kunne deretter ha lastet ned filen direkte.

Bruk av en mobil agent i søkeprosessen vil føre til en betydelig dårligere responstid ved søk. Dessuten er mobile agenter en sikkerhetstrussel, fordi personer med uhederlige hensikter kan misbruke slike agenter til å tilegne seg informasjon ulovlig.

En annen måte å bruke agenten på, er å la agenten lære hva brukeren liker. Agenten kan dermed bevege seg rundt i nettverket å finne musikk og videoer brukeren liker, uten at brukeren behøver å taste inn søkeord [14]. Agenten kan for eksempel lære om brukeren ved å studere materiale som brukeren tidligere har lastet ned eller gjennom hva slags type filer brukeren bruker mest.

7.9 Oppsummering

I dette kapittelet har vi diskutert de problemstillingene som oppgaven tar opp. Først så vi på P2P og P2P fildeling. Etter dette tok vi for oss JXTA og fordeler og ulemper med plattformen. Deretter sammenliknet vi JXTA med andre tilsvarende teknologier, Microsofts .net, Jini og Proem. Dagens versjon av JXME har en del begrensninger. Vi tok for oss forskjellene mellom dagens versjon av JXME og JXME 2.0.

Deretter diskuterte vi valgene som ble tatt i forbindelse med utviklingen av prototypen. Det ble også gjort beregninger av hvor mye det ville koste å bruke MOBster med dagens priser på datatrafikk. Oppgaven har handlet mye om begrensningene som finnes i mobiltelefoner. I avsnittet om mobilitet, tok vi frem noen aspekter ved mobilitet. I avsnittet relatert arbeide, så vi på Apeera. Kapittelet avsluttet med å se på mulige fremtidige scenarier for fildeling på mobiltelefoner.

I neste kapittel skal vi prøve å trekke noen konklusjoner ut av oppgaven samt lage en ”*roadmap*” for hva som må til for at fildeling på mobiltelefoner skal kunne slå an.

8 Konklusjon

I denne oppgaven har jeg vist at det er mulig å realisere P2P fildeling for mobiltelefoner. Med MOBster kan brukere hvor som helst i verden dele filer med hverandre gjennom JXTA nettverket. Dette er noe helt nytt for mobiltelefoner. Jeg har kommet med forslag til arkitektur for en P2P fildelingsapplikasjon på mobiltelefoner og laget metoder for å gjøre dagens implementasjon av applikasjonen mer effektiv (Eksempel: "Algoritme for beregning av antall poll", avsnitt 7.4.3.1). På grunn av tid og begrensninger i JXTA-plattformen ble det ikke anledning til å implementere alle kravene i kravspesifikasjonen (se vedlegg I). Per dags dato er det vanskelig å se noen praktisk nytte av en slik applikasjon på grunn av begrensningene i mobiltelefoner og JXME.

Med utgangspunkt i prototypen har vi i denne oppgaven sett på hvilke muligheter som ligger i teknologien. Vi har sett på begrensningene i form av ytelse, pris og lagringskapasitet i dagens teknologi, og litt på hvilke muligheter som ligger i morgendagens teknologi. Vi har også sett på mulige bruksområder for fildelingsapplikasjoner på mobiltelefoner. Ut i fra kunnskapen vi har tilegnet oss, vil vi prøve å lage en "*roadmap*", eller veibeskrivelse som sier litt om forutsetningene som må være på plass for at P2P-fildeling på mobiltelefoner skal kunne bli en virkelighet.

8.1 "Roadmap"

Vi mener at potensialet for P2P fildeling på mobiltelefoner er stort. Utbredelsen av mobiltelefoner med støtte for Java er stor og vil bare øke med årene [37]. Fildeling er en velkjent og populær applikasjon i stasjonære omgivelser. At brukere er vant til å bruke fildeling fra før, vil gjøre det enklere å ta i bruk fildelingsapplikasjoner på mobiltelefoner. Fildeling kan være med på å gi økt nytteverdi for mobiltelefonbrukere samtidig som det genererer mer trafikk for nettverksoperatørene.

Forskjellen mellom PDA-er og mobiltelefoner blir mindre og mindre. Sony Ericssons modell P800 er et eksempel på dette (se Figur 8-1). PDA-er har mulighet til å ringe, og mobiltelefoner får PDA-liknende funksjonalitet. De to terminaltypene er i ferd med å smelte sammen.

Figur 8-1 Sony Ericsson P800 (fra www.sonyericsson.com)

Vi har tidligere i rapporten argumentert for at mange av begrensningene i dagens versjon av MOBster kan omgås med ny versjon av MIDP (versjon 2.0) og JXME. Med mulighet for å "*pushe*" meldinger ut til mobiltelefoner, vil det ikke lenger være nødvendig for en peer å polle relèet etter innkommende meldinger. Hvis mobiltelefonene nærmer seg mer og mer PDA-er, og bare blir kraftigere og kraftigere, hva er da vitsen med å bruke MIDP i det hele tatt? Det er kun et tidsspørsmål før mobiltelefoner er like ressurssterke som dagens PDA-er. I løpet av få år vil mobiltelefoner uten problemer kunne kjøre CDC med for eksempel J2ME Personal Profile (PP). Med CDC og PP kan mobiltelefoner kjøre fullversjon av JXTA på egenhånd, det vil si at de kan kommunisere med JXTA-nettverket uten å gå via et relè. Hele problemstillingen med begrensninger i MIDP og JXME blir da uvesentlig.

Med CDC og PP vil mobiltelefonen kunne delta i JXTA-nettverket på lik linje med andre peer-er. Mobiltelefonen vil også kunne kommunisere med CMS på JXTA-peer-ene. Dette vil gi mobiltelefonene mulighet til å dele filer med alle peer-er i nettverket. Mobiltelefonens tilgang til informasjon vil øke og nytten av en fildelingsapplikasjon vil bli større.

Til tross for at mobiltelefonen ikke lenger er avhengig av relèet, kan likevel noen av de tjenestene relèet tilbyr være nyttige for mobiltelefonen. Selv om mobiltelefonen får større minne og prosessorkraft, er det ikke sikkert båndbredden til mobiltelefonen øker med det første. Prisen på datatrafikk er også uviss. UMTS og 3G lover båndbredde opp mot 2Mbit/s [66]. I virkeligheten vil det trolig dreie seg om maksimalt 384 kbit/s i nedlastning og 64 kbit/s i opplastning. Vi vet ennå ikke når startskuddet for UMTS og UMTS-terminaler går i Norge. Operatørene vil drøye lengst mulig for å få mest mulig igjen for investeringene i GSM-nettet. Det er heller ikke sikkert mobiltelefoner ønsker å motta absolutt alle meldinger som sendes til den. Et relè som filtrerer bort unødvendige meldinger og trimmer disse kan derfor være ønskelig. Mobiltelefonen vil fortsatt ha begrenset med batterikapasitet. Det er ingen vits i å overbelaste mobiltelefonen med mer trafikk enn nødvendig.

Om få år kommer kombinasjonsterminaler med WLAN og GSM/UMTS integrert i samme terminal. Vi tror IP-telefoni vil være godt utbredt med både UMTS og WLAN som bærere. WLAN tilbyr allerede hastigheter opp mot 54 Mbit/s[43]. WLAN-soner, såkalte "*HotSpots*", dukker opp over hele landet. Nylig besluttet Statoil å tilby trådløst Internett på alle sine bensinstasjoner. Et annet firma planlegger å bygge ut WLAN-soner i norske havner. Med godt utbygde WLAN-soner vil mobiltelefoner med WLAN-aksess kunne få tilgang til høy båndbredde til en lav pris. Høy båndbredde med lave kostnader er en forutsetning for at P2P fildeling for mobiltelefoner skal lykkes.

Et fjerde generasjons mobiltelefonsystem (4G) er under planlegging. 4G skal tilby båndbredde opp mot 100Mbit/s eller mer [48]. 4G er planlagt ferdig i 2010. Mange stiller seg spørsmål om nytten av et slikt nett, fordi WLAN om få år vil kunne tilby liknende båndbredde.

8.2 Fremtidig arbeid

I løpet av arbeidet med oppgaven har jeg kommet med forslag til områder som det bør arbeides mer med om en P2P fildelingsapplikasjon for mobiltelefoner skal kunne bli en realitet. Under er en oversikt over forslag til fremtidig arbeid:

- Avgrensning av søk. Det finnes ingen måter for å avgrense søk i MOBster. I oppgaven er det kommet forslag til mulige metoder som kan benyttes for å avgrense søkene. En metode for å avgrense søk bør implementeres i applikasjonen (se avsnitt 7.4.4).
- Realisere MOBster som en *Applet*. Applet-en fungerer som en mellomting mellom en full JXTA-peer og en JXME-peer. En Applet vil kunne fungere som et mellomstadie på veien til realisering av fildeling mellom JXTA-peer-er og JXME-peer-er (se avsnitt 6.1.5).

[43] WLAN kommer i tre utgaver. IEEE 802.11a (54Mbit/s), IEEE 802.11b (11Mbit/s) og IEEE 802.11g (22Mbit/s)

- Realisere fildeling mellom JXTA-peer-er og JXME-peer-er. Dette er beskrevet i detalj i scenario tre, avsnitt 6.1.3.
- Overføre filer med BlueTooth. Mobiltelefoner som befinner seg i geografisk nærhet, kan velge å laste ned filene direkte fra hverandre via BlueTooth. Dette er beskrevet i scenario fire, avsnitt 6.1.4.
- Utvikle metoder for å lagre data på eksternt lagringsmedium (avsnitt 7.8.1).
- Se på nærmere på om en **mobil agent** (avsnitt 7.8.3) har noe for seg for P2P fildelingsapplikasjoner for mobiltelefoner.

Amish-folket i USA er kjent for å leve gammeldags. Det er ikke alt Amish-folket gjør som er like fornuftig, men de har en veldig sunn holdning til ny teknologi. Er all ny teknologi god for oss? For hver ny teknologi som kommer stiller de seg spørsmålet ”Er dette noe som vil bringe oss nærmere hverandre?” [35]. Dette er årsaken til at Amish-folket blant annet ikke har telefon i husene sine. De mener telefonen ødelegger for samværet i hjemmet. Vil P2P-fildeling på mobiltelefoner bringe oss nærmere hverandre?

Referanser

[1] A. Arora, C. Haywood, K.S. Pabla, *JXTA for J2ME – Extending the reach of Wireless With JXTA Technology*, 2002
http://jxme.jxta.org

[2] A. Singla and C. Rohrs, Ultrapeers: Another Step Towards Gnutella Scalability, Dec 2001
http://RFC-gnutella.sourceforge.net/Proposals/Ultrapeer/

[3] A. Zieger, *Relating peer to peer: Developing implementation that lets all points wireless get along*¸ IBM Developerworks, oct 2001
http://www-10.ibm.com/developerworks/

[4] B. Wilson, *JXTA book,* sist besøkt desember 2002
http://www.brendonwilson.com/projects/jxta/index.shtml

[5] C. Freeman, C. Perez, *Structural crisis of adjustment, buisness cycles and investment behavior*, i G.Dosi et al (red) Technical change and economic theory, London: Printer Press, 1988, s. 38-66

[6] C. Lindemann and O.P. Waldhorst, *A Distributed Search Service for Peer-to-Peer File Sharing in Mobile Applications*, Proc. 2nd IEEE Conf. on Peer-to-Peer Computing (P2P 2002), Linköping, Sweden, September 2002

[7] C. Shirkly, *Listening to Napster,* Harnessing the Power of Disruptive Technologies, ISBN: 0-596-00110-X, 2001

[8] C. Vawter and E. Roman, *J2EE vs. Microsoft.NET: A comparison of building XML-based web services*, 2001
http://www.theserverside.com/resources/article.jsp?l=J2EE-vs-DOTNET

[9] D. Briklin, *The Cornucopia of the commons*, Harnessing the Power of Disruptive Technologies, ISBN: 0-596-00110-X, 2001

[10] D. Isenberg, *Rise of the Stupid Network*, ACM Networker 2.1, February/March 1998

[11] E. Giguere, *Understanding J2ME Application Models*, 2002
http://wireless.java.sun.com

[12] *Efficient MIDP Programming*, Forum Nokia, sist besøkt mars 2003
http://forum.nokia.com

[13] G. Kan, Gnutella and GoneSilent.com, *Gnutella*, Harnessing the Power of Disruptive Technologies, ISBN: 0-596-00110-X, 2001

[14] G. Korteum, J. Schneider, D. Preuitt, T.G.C. Thompson, S. Fickas and Z. Segall, *When Peer-to-Peer comes Face-to-Face: Collaborative Peer-to-Peer Computing in Mobile Ad hoc Networks*, proceedings 2001 International Conference on Peer-to-Peer Computing, Linkøping, Aug 27-29 2001

[15] G. Søderstrøm, *The JXTA protocol for mobile devices*, Telia Research AB, July 2002

[16] G.Coulouris, J.Dollimore, T.Kindberg, *Distrbuted Systems: Concepts and Design 3rd edition*, ISBN 0201-61918-0, 2001

[17] *Gnutella vs. JXTA*, CM316: Multimedia Systems Coursework, 2001

[18] J. Hjelm, *Designing Wireless Information Services*, ISBN: 0-471-38015-6, 2000

[19] J. Knudsen, *What's New in MIDP 2.0*, 2002
http://wireless.java.sun.com

[20] J.S. Brown and P. Duguid, *Borderline issues: social and material aspects of design*, Human Computer Interaction 9: 3-36, 1994

[21] L. Capra, W. Emmerich and C. Mascolo, *Middleware for Mobile Computing: Awarenes vs. Transparency*, Position Summary. In Proc. of the 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII). Schloss Elmau, Germany. May 2001.

[22] L. Gong, JXTA: *A Network Programming Environment*, IEEE Internet Computing, may-june 2001

[23] L. Gong, *Project JXTA: A technology Overview*, 2001
http://www.jxta.org

[24] L. Gutberlet, *Peer-to-Peer Computing – A Technology Fad or Fact?*, Information Systems Management Seminar WS 2000, Oct 2000

[25] M.J. Yuan, *Mobile P2P messaging, Part 2: Develop mobile extensions to generic P2P networks*¸ IBM Developerworks, 2003
http://www-10.ibm.com/developerworks/

[26] N. Minar, M. Hedlund, *A Network of Peers: Peer-to-Peer Models Through the History of the Internet*, Harnessing the Power of Disruptive Technologies, ISBN: 0-596-00110-X, 2001

[27] P. Agre, *Changing Places: Context of Awareness in Computing*, 2001
http://dlis.gseis.ucla.edu/pagre/

[28] P. Lamsal, *J2ME Arcitecture and Related Embedded Technologies*, 2001
http://www.cs.helsinki.fi/u/campa/teaching/j2me/papers/J2ME.pdf

[29] P. Luff and C. Heath, *Mobility in Collaboration*, In Computer Supported Collaborative Work Conference (CSCW), Seattle, Washington, 1998

[30] *Peer-to-Peer File Sharing: The impact of file sharing on service provider networks*, Sandvine, Sept 2002
http://www.sandvine.com/solutions/p2p_policy_mngmt.asp

[31] Q. Mahmoud, *Deploying Wireless Java Applications*, 2002
http://wireless.java.sun.com

[32] Q. Mahmoud, *Secure Java MIDP Programming Using HTTPS with MIDP*, 2002
http://wireless.java.sun.com

[33] Q. Mahmoud, *Wireless Java™ Security*, 2002
http://wireless.java.sun.com

[34] Q. Mahmoud, *Wireless Software Design Techniques: "What every wireless software developer should know"*, 2002
http://wireless.java.sun.com

[35] Rheingold, *Look who's talking*, Wired Magazine 7-01, 1999
http://www.wired.com/wired/archive/7.01/amish.html

[36] S. Li, *The JXTA Story*, IBM Developerworks, 2001
http://www-10.ibm.com/developerworks/

[37] S. MCAteer, *Java Will Be the Dominant Handset Platform*, Micro Java Network, 2002
http://www.microjava.com/articles/perspective/zelos

[38] S.S Bygdås, R.H. Johansen, Ø. Myhre, T. Urnes, *Java Applications on Wireless Information Devices*, R&D, 9/2001, ISBN: 82-423-0393-2, 2001

[39] T. Huang, *Digital Signatures on Mobile Devices – Reality Check*, 2002
http://www.wirelessdevnet.com

[40] *The Battle Over Mobile Platform Software: The Center of Gravity Shifts: Java is the Platform to Target*, Yahoo! Finance, 2003
http://biz.yahoo.com

[41] V. Belotti and S.Bly, *Walking Away from the Desktop Computer: Distributed Collaboration and Mobility in a Product Design Team*, Computer Supported CooperativeWork, ACM, 1996

[42] W. Stallings, *Cryptography and Network Security: Principles and Practice*, ISBN 0-13-869017-0, 1998

[43] .net, sist besøkt mars 2003
http://www.microsoft.com/net

[44] *"Mobile Internett er ikke dødt, det trenger bare litt assistanse"*, ITavisen, 2002
http://telecom.no/arkiv/art/4956.html

[45] *"Rekordøkning for bredbånd"*, Dagbladet, mars 2003
http://www.dagbladet.no/dinside/2003/03/18/364264.html

[46] *"Mobiltelefoner og helseskader: Hva vet vi og hva så?"*, Teknologirådet, 2002
http://www.teknologiradet.no/

[47] *"Ni av ti unge sender SMS hver dag"*, VG-nett, sist besøkt januar 2003
http://www.vg.no/pub/vgart.hbs?artid=9719804

[48] A. Dornan, *Fast Forward to 4G?*, mars 2002
http://www.commweb.com/article/NMG20020304S0010

[49] Apeera, sist besøkt februar 2003
http://www.apeera.com

[50] Applets, sist besøkt mars 2003
http://java.sun.com/applets/

[51] BlueTooth, sist besøkt mars 2003,
https://www.bluetooth.org/

[52] Content Management System (CMS), sist besøkt mars 2003
http://cms.jxta.org

[53] Extensible Markup Language (XML), sist besøkt april 2003
http://www.w3.org/XML/

[54] Forum Nokia, sist besøkt mars 2003
http://www.forum.nokia.com/

[55] GPS World, sist besøkt april 2003
http://www.gpsworld.com/

[56] IPv6, Illustrert Vitenskap, november 2002
www.illustrertvitenskap.com/200211/7787.asp

[57] Java 2 Platform, Micro Edition, sist besøkt april 2003
http://java.sun.com/j2me

[58] Jini, sist besøkt mars 2003
http://www.jini.org/

[59] JXTA for J2ME (JXME), sist besøkt april 2003
http://jxme.jxta.org/

[60] KaZaA, sist besøkt april 2003
http://www.kazaa.com

[61] ***MIDP APIs for Wireless Applications*****, 2001**
http://java.sun.com/j2me

[62] Semantic Web, ***Peer-to-Peer: The Infrastructure for the Semantic Web*****, sist besøkt april 2003**
http://p2p.semanticweb.org/

[63] ***Programming Languages in Symbian*****, Symbian.com, 2003**
http://www.symbian.com/developer/

[64] ***Security and Project JXTA*****, sist besøkt desember 2002**
http://www.jxta.org

[65] Sony.com, ***Sony MemoryStick*****, sist besøkt april 2003**
http://www.ita.sel.sony.com/memorystick

[66] Universal Mobile Telecommunications System (UMTS), sist besøkt mars 2003
http://www.umts-forum.org

Bibliografi

A. Bateman, ***Digital Communications*****, ISBN 0-201-34301-0, 1999**

A. Oram, ***Peer-to-Peer: Harnessing the Power of Disruptive Technologies*****, ISBN 0-596-00110-X, 2001**

I. Horton, ***Beginning Java 2: A Comprehensive tutorial to Java programming*****, ISBN 1-861002-23-8, 1999**

I. Sommerville, ***Software Engineering 6th Edition*****, ISBN 0-201-39815-X, 2001**

J. Delebeke, ***Recent long-wave theories – a critical survey",*** **Futures, august 1981, s. 245-257**

M. Fowler and K. Scott, ***UML Distilled: A Brief Guide to the Standard Object Modeling Language*****, ISBN 0-201-65783-X, 2000**

M.T. Goodrich, ***Data Structures and Algorithms in Java*****, ISBN 0-471-19308-9, 1998**

O'Reilly Network, http://www.openp2p.com, 2003

Portable Network Graphics**, http://www.libpng.org/pub/png/, 2003**

R. Elmasri and S. B. Navathe, ***Fundamentals of Database Systems*****, ISBN 0-201-54263-3, 2000**

S. Mullender, ***Distributed Systems 2nd Edition*****, ISBN 0-201-62427-3, 1994**

S. S. Epp, ***Discrete Mathematics with Applications,*** **ISBN 0-534-94446-9, 1995**

S. Sen, J. Wang, ***Analyzing peer-to-peer traffic across large networks*****, submitted to ACM/IEEE Transactions on Networking (ToN), 2003**

Forkortelser

ADSL - Asymmetric Digital Subcriber Line
AMS - Application Management Software
API - Application Programming Interface
CA - Certificate Authority
CDC - Connected Device Configuration
CLDC - Connected Limited Device Configuration
CLR - Common Language Runtime
CMS - Content Manager Service
CPU - Central Processing Unit
DHCP - Dynamic Host Configuration Protocol
DNS - Domain Name System
FCL - Framework Class Library
FP - Foundation Profile
FTP - File Transfer Protocol
GPRS - General Packet Radio Service
GPS - Global Positioning System
HSCSD - High Speed Circuit Switched Data
HTTP - HyperText Markup Language
IL - Microsoft Intermediate Language
IM - instant messaging
IP - Internet Protocol
IR - InfraRed
J2EE - Java 2 Enterprise Edition
J2ME - Java 2 Micro Edition
J2SE - Java 2 Second Edition
J2WTK - Java 2 Micro Edition Wireless Toolkit
JAD - (Java Application Descriptor)
Java RMI - Java Remote Method Invocation
JDK - Java Development Kit
JVM - Java Virtual Machine
JXME - JXTA for J2ME
MAC - Medium Access Control
MANET - Mobile Ad-hoc NETwork
MIDP - Mobile Information Device Profile
MMS - Multimedia Messaging Service
MPEG - Moving Picture Experts Group
NAT - Network Address Translation
NTP - Network Time Protocol
OS - Operating System
PAN - Personal Area Network
PBP - Personal Basis Profile
PDA - Personal Digital Assistant
PDI - Passive Distributed Indexing
PNG - Portable Network Graphics
PP - Personal Profile
QoS - Quality of Service
RMI - Remote Method Invocation
RMS - Record Management System
SETI - Search for Extraterrestrial Intelligence
SMS - Short Message Service
SOAP - Simple Object Access Protocol
SSL - Secure Socket Layer
TCP - Transmission Control Protocol
TLS - Transport Layer Security

TTL - Time To Live
UDDI - Universal Description, Discovery and Integration
UDP - User Datagram Protocol
UML - Unified Modeling Language
UMTS - Universal Mobile Telecommunications System
URI - Uniform Resource Identifier
URL - Uniform Resource Locator
URN - Uniform Resource Name
UUID - Unique Universal IDentifier
VB - Visual Basic
WAP - Wireless Application Protocol
WAV - Waveform audio file
WLAN - Wireless Local Area Network
WSDL - Web Service Definition Language
WWW - World Wide Web
XML - Extensible Markup Language

Vedlegg

I. Kravspesifikasjon

Det ble utarbeidet to kravspesifikasjoner for MOBster: en formell, og en uformell. Den ferdige prototypen avviker fra kravspesifikasjonene. En årsak til dette er at teknologien som ble benyttet i prosjektet foreløpig er umoden. Mye uforutsette ting har kommet i veien.

Uformell kravspesifikasjon

Innledning

Deltakere

Hovedfagsstudent:	Henrik Heiestad	(UiO, institutt for informatikk)
Ekstern veileder:	Sigrid Steinholt Bygdås	(Telenor FoU)
Intern veileder:	Kjell Åge Bringsrud	(UiO, institutt for informatikk)

Formål

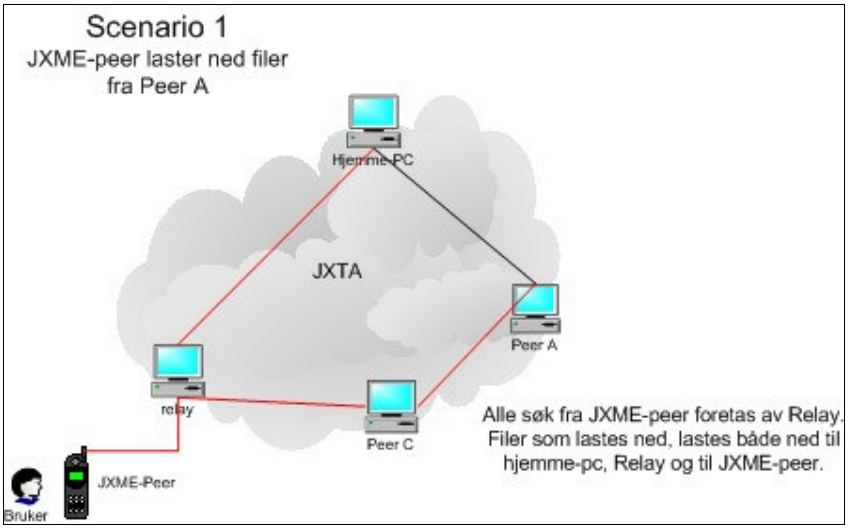
Bakgrunn
Systemet er en del av Henrik Heiestad sin hovedoppgave ved Institutt for informatikk. Hovedoppgaven har som mål "å realisere P2P fildeling mellom mobiltelefoner". **P2P fildeling** defineres her som at medlemmer (noder, peer-er) i et spontant anarkistisk nettverk skal kunne:

- gjøre egne filer tilgjengelige for andre
- søke etter filer av interesse
- laste ned filer fra andre

Systemet skal være med på å gi svar på en del problemstillinger i hovedoppgaven:

- Er det mulig å lage en brukbar fildelingsapplikasjon for en så begrenset terminal som en mobiltelefon? Er det mulig å overføre fildelingskonseptet for stasjonære terminaler over på mobiltelefoner?
- Hvilke bruksområder og brukssituasjoner ser vi for oss? Hvilke type filer ønsker brukere å dele?
- Hva har systemet å si for brukerens mobilitet? Hvilken effekt har systemet? Hvordan kan det endre brukers adferd?

Teknologien for å realisere **mobster** vil være JXTA for J2ME (JXME).



Figur I.1 Ett mulig scenario for mobster.

Programmet skal:

- kunne søke etter filer i JXTA-nettverket.
- kunne laste ned filer fra andre JXTA-peer-er.
  med filer menes tekst, bilder, videoklipp, musikk, ringetoner, logoer, spill.
- kunne vise filer som er lagret.
  med filer menes filer som ligger lagret lokalt på mobiltelefonen. (disse kan ikke deles med andre, siden J2ME ikke har støtte for CMS (Content Management System).
- kunne endre egne innstillinger
  med egne innstillinger menes brukernavn, passord, gruppe etc.
- kunne se generell info om programmet

Programmet bør:

- kunne se egne filer (distribuert filsystem).
  med egne filer menes her filer som ligger lagret på alle dine maskiner (hjemme-pc, jobb-pc, PDA).
- gjøre egne filer tilgjengelige for andre
  med dette menes at du skal kunne gjøre filer som ligger lagret på dine maskiner (hjemme-pc, jobb-pc, PDA) tilgjengelige for andre JXTA-peer-er.
- kunne laste ned filer fra andre JXTA-peer-er til dine maskiner (hjemme-pc, jobb-pc, PDA).
- kunne se status på nedlastninger.
- kunne gi støtte for økt sikkerhet ved hjelp av kSSL (mini-versjon av SSL, Secure Socket Layer, som benyttes i HTTPS)

Det vil trolig bli en ekstra utfordring å få koblet seg opp mot jobb-pc, fordi denne som regel er beskyttet av forskjellige sikkerhetsløsninger. Det ligger utenfor oppgavens problemområde å finne løsninger på dette. Oppgaven forutsetter at det går an å oppnå kontakt med alle maskiner som er tilknyttet JXTA-nettverket.

Utvikling

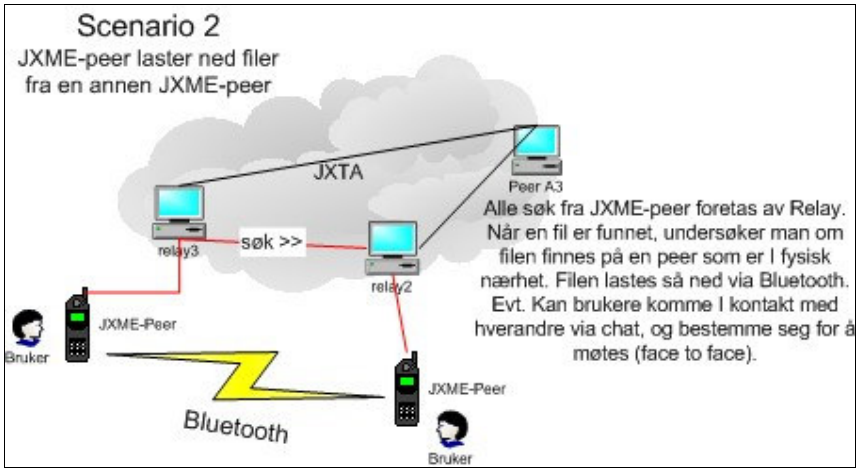
Det er meningen at **mobster** skal være en del av en programpakke bestående av tre applikasjoner, Chat, fildeling og MP3-avspiller.

Tidsfrister

1. juni 2002	Kravspesifikasjon skal være ferdig
23. juni 2002	En begrenset prototyp skal være utviklet

Grensesnitt

Internt
Filer som er lastet ned skal kunne deles med andre applikasjoner innenfor samme MIDlet suite (jfr. MIDP 1.0).



Figur I.2 Nedlastning av filer over Bluetooth

Eksternt
MOBster skal være implementert slik at brukere kan søke etter filer hos og laste ned filer fra alle JXTA-peer-er, forutsatt at peer kjører CMS (Content Management System). Hvis mulig skal mobster kunne foreta søk via JXTA-nettverket, og sjekke om noen av peer-ene som har de ønskede filene befinner seg i nærheten og har støtte for Bluetooth. Bluetooth-kommunikasjonen skal da settes opp direkte mellom de to peer-ene og filene lastes ned uten å benytte GPRS/UMTS. (Se Figur I.2 Nedlastning av filer over Bluetooth)

Funksjonelle egenskaper

Overordnet systembeskrivelse
System: P2P-fildelingsapplikasjon for mobiltelefon. Systemet benevnes **mobster** (En sammenslåing av ordene MOBiltelefon og napSTER).
Bruker: en som benytter **mobster**.
Peer: en vilkårlig peer i JXTA-nettverket.

(Benytter IEEE/ANSI 830-1993 for å klassifisere systemkravene)

Klassifisering	Id	Beskrivelse	Prioritet	Akseptansetest
Brukergrensesnitt	B.1	Bruker skal kunne logge seg inn med brukernavn og passord	HØY	Testes ved å taste inn brukernavn som ikke er gyldige (tomt brukernavn og passord)
Brukergrensesnitt	B.2	Bruker skal kunne endre innstillinger, relè, brukernavn, passord.	HØY	Testes ved å forandre innstillinger.
Brukergrensesnitt	B.8	Kunne administrere egne filer på andre maskiner (hjemme-pc etc.). Slette, gi rettigheter.	MIDDELS	

Brukergrensesnitt	B.9	Kunne hente filer fra hjemme-pc.	MIDDELS	
Brukergrensesnitt	B.3	Bruker skal kunne foreta søk etter en gitt filtype, og få presentert resultatet på en oversiktlig måte.	HØY	
Brukergrensesnitt	B.4	Bruker skal kunne laste ned valgt fil. Filen som lastes ned skal kunne lastes ned både til hjemme-pc og til mobiltelefon. (Valgfritt om bruker vil laste til begge eller bare til en av de to)	HØY	
Brukergrensesnitt	B.5	Bruker skal ha muligheten til å se status på nedlastning, og evt. avbryte nedlastningen.	MIDDELS	
Brukergrensesnitt	B.6	Bruker skal kunne se egne filer (filer som befinner seg på mobiltelefonen) og slette egne filer.	HØY	
Brukergrensesnitt	B.7	Bruker skal kunne bytte gruppe og opprette nye grupper.	MIDDELS	

Ikke-funksjonelle egenskaper

Klassifisering	Id	Beskrivelse	Prioritet	Akseptansetest
System	S.1	Søkene skal foretas via relè/*proxy* ut i JXTA-nettverket.	HØY	Testes ved å forsøke og finne filer på andre maskiner enn de internt på LAB-nettet til Telenor FoU.
System	S.2	Det skal være mulig å dele filer med andre JXTA-peer-er over Bluetooth (direkte kommunikasjon)	LAV	
Lagring	L.1	Filstørrelser må ikke overskride tilgjengelig lagringskapasitet på mobiltelefon	MIDDELS	Testes ved å forsøke å laste ned alt for store filer til mobiltelefonen.
Lagring	L.2	Hvis mulig skal filer som lagres lokalt på mobiltelefon overføres ved hjelp av Bluetooth til en Bluetooth lommeserver.	LAV	

Informasjonssikkerhet

All informasjon som lagres i MOBster er kun tilgjengelig for applikasjoner innenfor samme MIDlet suite (det vil si Chat-applikasjon, fildelingsapplikasjonen og MP3-avspiller). Dette gjør at det ikke er mulig for andre programmer å få tilgang til informasjonen lagret i **mobster** (filer og innstillinger). Dette er noe som er gjort i J2ME for å hindre at programmer skal få tilgang til dine personlige opplysninger (telefonkatalog, kontaktinformasjon m.m.) og tekstmeldinger.

Testprosedyrer

Utviklingen av systemet skal skje på emulatorer (J2ME Wireless Toolkit). Alle moduler testes kontinuerlig under utvikling. Når systemet er ferdig utviklet vil det bli gjort systemstest av veileder og student.

Det finnes allerede terminaler på markedet (blant annet Nokia 9210 Communicator, Nokia 3410, Nokia 6310i) som støtter J2ME (MIDP 1.0 og CLDC). Hvis mulighetene byr seg, vil det være ønskelig å foreta tester på en slik terminal (eller nye terminaler som støtter J2ME). Dette vil være spesielt viktig for å teste hvordan programmet påvirker brukers mobilitet og programmets effekt på adferd.

Formell kravspesifikasjon

Innledning

mobster inngår som en del av hovedoppgaven til Henrik Heiestad ved Universitet i Oslo. mobster er en fildelingsapplikasjon som sammen med en chat-applikasjon og en MP3-spiller skal utgjøre en programpakke (en MIDlet suite). (les mer om bakgrunnen for mobster i ”uformell kravspesifikasjon v1.1”)

Under spesifiseres aktørene i systemet:

Aktører

Aktør	Bruker
Beskrivelse	Bruker er bruker av systemet. Bruker kan søke etter filer, laste ned filer og ”dele” filer.
Eksempler	Ungdom, studenter, menn, kvinner, eldre (alle med mobiltelefon som støtter Java (J2ME))

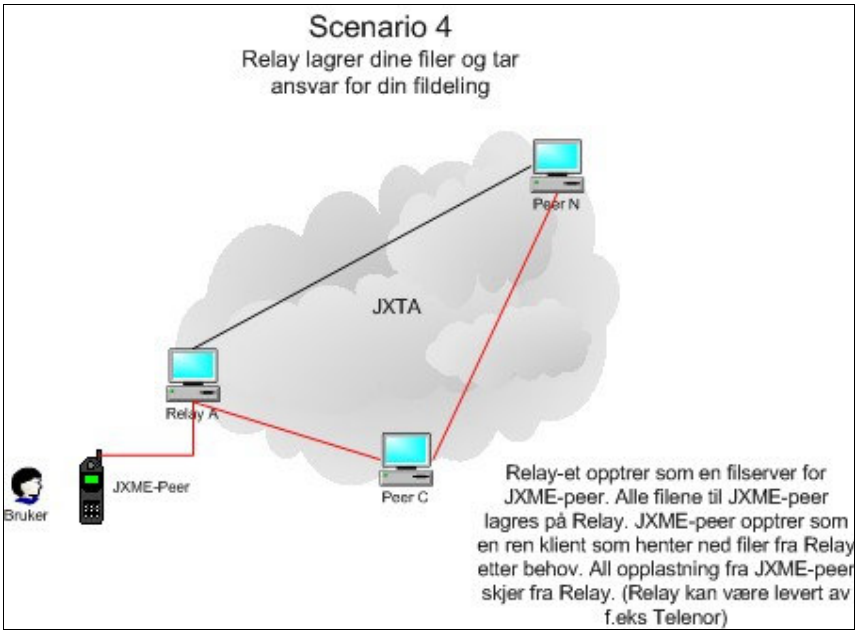
Tabell I.2 Aktør bruker

Aktør	ISP (Internet Service Provider).
Beskrivelse	Tilbyr relè-tjenester og fil-lagringstjenester for mobiltelefoner. Ingen aktiv bruker av systemet.
Eksempler	Telenor Mobil

Tabell I.3 Aktør ISP

ISP (Internet Service Provider)

Et av scenariene vi har sett for oss, inneholder en ISP (Internet Service Provider). Scenariet beskriver hvordan en slik ISP tilbyr lagringsplass og relè-tjenester for JXME-peer-er. Alle filene til bruker (JXME-peer) blir lagret på relè. Brukeren kan selv administrere disse filene. Når bruker ønsker det kan han/hun laste ned filer til sin mobiltelefon og spille av disse. Det er relè som tar seg av opplastninger som initieres fra andre JXTA-peer-er (siden filene er fysisk plassert hos relè). (Se Figur I.2: Scenario med ISP)



Figur I.2: Scenario med ISP

Bruksmønster

I dette delkapittelet spesifiseres bruksmønstrene for systemet. Figur I.3 viser relasjonene mellom aktørene og de ulike bruksmønstrene. Bruksmønstrene beskrives på de følgende sidene.

Bruksmønster	Innlogging
Aktør	Bruker
Trigger	Bruker ønsker å benytte mobster
Pre-betingelser	1) Mobiltelefonen har tilgang til nettet. 2) mobster er riktig installert.
Post-betingelser	1) Bruker er logget inn og kan benytte mobster. 2) Bruker er ikke logget inn og har fått tilbakemelding om hva som gikk feil.
Normal hendelsesflyt	1) Bruker skrur på systemet 2) Systemet ber om brukernavn og passord 3) Bruker taster inn brukernavn og passord 4) Systemet verifiserer brukernavn og passord 5) Systemet sjekker om den oppnår kontakt med JXTA-nettverket. 6) Systemet gir melding om at innlogging var vellykket.
Variasjoner	4a) Systemet oppdager uoverensstemmelser mellom brukernavn og passord. 4b) Systemet gir feilmelding og går til punkt 2). Ved femte gale forsøk, sperres systemet for flere forsøk. 5a) Systemet klarer ikke å opprette kontakt med JXTA-nettverket. 5b) Systemet gir feilmelding og viser vanlig skjermbilde (bruker har mulighet til å se på egne filer). Knapp for å forsøke å koble seg til på nytt må være tilgjengelig. (eventuelt kan det ligge en funksjon i systemet, som automatisk forsøker å gjenopprette forbindelsen til JXTA-nettverket)
Relatert informasjon	Det er mulig vi kan legge det opp slik at brukere slipper å autentisere seg når de bruker mobster (annet enn første gang). Det kan være nok at bruker klarer å slå på telefonen (PIN-kode), evt at han slår en pin-kode for mobster og.

Tabell I.4 Bruksmønster innlogging

Bruksmønster	Endre innstillinger
Aktør	Bruker
Trigger	Bruker ønsker å endre innstillinger
Pre-betingelser	1) Bruker er innlogget
Post-betingelser	1) Bruker har endret innstillingene 2) Bruker har avbrutt endring av innstillingene
Normal hendelsesflyt	1) Systemet viser alle lagrede innstillinger 2) Bruker velger innstilling han/hun ønsker å endre. 3) Bruker endrer innstillingen og lagrer. 4) Systemet verifiserer at endringen er gyldig 5) Systemet lagrer endringer
Variasjoner	4a) Systemet finner feil i formatet på input fra bruker. (Med feil i formatet menes at det tastes karakterer i stedet for heltall i IP-adresser for eksempel). Systemet ber bruker taste inn informasjon på nytt.
Relatert informasjon	Innstillinger vil bli: Brukernavn, passord, alias (brukernavn), relè, port, poll (hvor ofte mobster skal kontakte relè for å se etter innkommende meldinger), antall samtidige nedlastninger som er tillatt, hjemmemaskin, jobb-maskin +++ Det må hele tiden være mulig å avbryte funksjonen.

Tabell I.5 Bruksmønster endre innstillinger

Bruksmønster	Søk
Aktør	Bruker
Trigger	Bruker ønsker å finne en fil i JXTA-nettverket
Pre-betingelser	1) Bruker er innlogget og tilkoblet JXTA-nettverket
Post-betingelser	1) Bruker har funnet filen han/hun søker etter 2) Filen ble ikke funnet i JXTA-nettverket 3) Bruker har avbrutt søket
Normal hendelsesflyt	1) Bruker taster inn navn på filen han/hun leter etter 2) Bruker velger filtype (evt. alle typer) Filtyper kan være spesifisert i en *combobox*. Nye filtyper blir lagt til etterhvert som bruker taster de inn. (Dette må da lagres av systemet for hver gang et søk foretas) 3) Systemet foretar søk via relè 4) Systemet presenterer resultatet
Variasjoner	1a) Bruker taster inn navn på filen med * og ? som ”*wildcars*” 3a) Systemet oppnår ikke kontakt med relè. Feilmelding skrives ut. Søk avsluttes. Bruker må få muligheten til å forsøke å koble seg opp på nytt.
Relatert informasjon	Det må eksistere noen form for innstillinger her og. Bruker må kunne velge hvilken informasjon om filene han/hun ønsker å få presentert. I tillegg må bruker få må mulighet til å avgrense hvor mange treff han/hun ønsker å se.

Tabell I.6 Bruksmønster søk

Bruksmønster	Last ned
Aktør	Bruker
Trigger	Bruker ønsker å laste ned en fil han har funnet
Pre-betingelser	1) Bruker har funnet en fil gjennom søk, som han ønsker å laste ned 2) Bruker ønsker å laste ned en av filene på sin hjemme-pc
Post-betingelser	1) Filen er lastet ned på mobiltelefon og hjemme-pc (valgfritt hvor det skal lastes ned) 2) En feil oppstod under nedlastning. Filen ble ikke lastet ned. 3) Bruker har avbrutt nedlastning.
Normal hendelsesflyt	1) Bruker markerer fil han/hun ønsker å laste ned 2) Bruker velger funksjon for å laste ned 3) Systemet initierer nedlastning til terminal via relè 4) Systemet verifiserer at det er nok lagringsplass på terminal 5) Systemet initierer nedlastning til hjemme-pc 6) Systemet laster ned filen 7) Systemet lagrer filen direkte i RMS
Variasjoner	3a) Systemet får feilmelding tilbake fra relè. Nedlastning avbrytes. 4a) Filen er for stor. Nedlastning avbrytes. Feilmelding vises på skjermen. 6a) Nedlastning avbrytes fordi forbindelsen til JXTA-nettverket går ned. Feilmelding vises. Bruker får mulighet til å koble seg til JXTA-nettverket på nytt. 6b) Nedlastning avbrytes fordi peer-en brukeren laster ned fra avbryter nedlastning. Feilmelding vises. Bruker kan foreta ny nedlastning.
Relatert informasjon	Det må eksistere noen form for innstillinger her og. Bruker må kunne velge om filen skal lastes ned til hjemme-pc (relè) eller ikke. Viktig at dette blir enkelt.

Tabell I.7 Bruksmønster last ned

Bruksmønster	Se status (på nedlastninger)
Aktør	Bruker
Trigger	Bruker ønsker å se status på nedlastning
Pre-betingelser	1) Systemet har startet nedlastning av en fil
Post-betingelser	1) Bruker har sett status på nedlastning 2) Bruker har avbrutt nedlastning
Normal hendelsesflyt	1) Bruker velger funksjon for å se status 2) Systemet presenterer status for nedlastning (med hastighet og tid igjen)
Variasjoner	
Relatert informasjon	Dette bruksmønsteret er valgfritt (ingen viktig funksjon for systemet)

Tabell I.8 Bruksmønster se status

Bruksmønster	Se filer
Aktør	Bruker
Trigger	Bruker ønsker å se egne filer
Pre-betingelser	1) Bruker er logget på
Post-betingelser	1) Bruker har sett egne filer
Normal hendelsesflyt	1) Systemet henter informasjon om filer som ligger lagret lokalt. 2) Systemet kobler seg mot relè og innhenter informasjon om alle filer som tilhører gruppen til bruker. Med gruppe menes en gruppe som brukeren oppretter hvor alle brukerens maskiner er deltakere. 3) Systemet presenterer filene. (Filene som er lagret lokalt må merkes på en eller annen måte)
Variasjoner	2a) Systemet klarer ikke å koble seg opp mot hjemme-pc eller andre maskiner som tilhører gruppa til brukeren. Systemet viser bare filer som er lagret lokalt. Feilmelding vises. Bruker har mulighet til å forsøke å koble seg opp mot JXTA-nettverk på nytt.
Relatert informasjon	Her må det være en rekke innstillinger. Bruker må kunne velge hvor mye info som skal vises om hver enkelt fil. Bruker må kunne velge om han bare skal se filer som er lagret lokalt.

Tabell I.9 Bruksmønster se filer

Bruksmønster	Endre filer
Aktør	Bruker
Trigger	Bruker ønsker å endre egne filer (gi nytt navn, flytte, slette)
Pre-betingelser	1) Bruker er logget på 2) Bruker har valgt en fil
Post-betingelser	1) Bruker har endret egen fil 2) Bruker har avbrutt endring
Normal hendelsesflyt	1) Systemet presenterer all info om filen 2) Bruker endrer det han ønsker 3) Systemet verifiserer det bruker har endret 4) Systemet ber brukeren bekrefte endringen 5) Systemet lagrer endringer
Variasjoner	3a) Systemet oppdager feil. Systemet gir feilmelding. (Feil kan være: for langt filnavn, filnavnet er tomt, ugyldig filnavn...) 4a) Bruker avbryter. Bruksmønster avsluttes.
Relatert informasjon	Med endring menes her i hovedsak muligheten til å slette, flytte, gi nytt navn eller gi rettigheter til en fil.

Tabell I.10 Bruksmønster endre filer

Bruksmønster	Logg ut
Aktør	Bruker
Trigger	Bruker ønsker å logge ut
Pre-betingelser	1) Bruker er logget på
Post-betingelser	1) Bruker er logget ut 2) Bruker avbrøt utlogging
Normal hendelsesflyt	1) Systemet spør om bruker virkelig vil logge ut 2) Systemet lagrer all informasjon og avslutter programmet
Variasjoner	1a) Bruker avbryter utlogging.
Relatert informasjon	Når bruker er logget ut, avsluttes mobster. Bruker kan da starte opp andre applikasjoner (chat, MP3)

Tabell I.11 Bruksmønster logg ut

Kommentar

Det er mulig det blir flere bruksmønstre etterhvert, litt avhengig av hvilke behov vi ser underveis i prosjektet. Prosjektet benytter en inkrementell utviklingsmodell. M**obster** vil i hovedsak konsentrere seg om de funksjonelle egenskapene beskrevet i uformell kravspesifikasjon, med prioritet HØY og MIDDELS.

Det er forholdsvis mange kompliserte funksjoner som er spesifisert i systemet. Det vil bli lagt mye arbeid i å gjøre systemet enkelt å bruke. Antakeligvis vil brukere måtte gjøre litt arbeid første gangen de bruker programmet, men etter det vil bruken bli mye enklere. Det er viktig å finne riktig balanse mellom fleksibilitet og administrativ *overhead*.

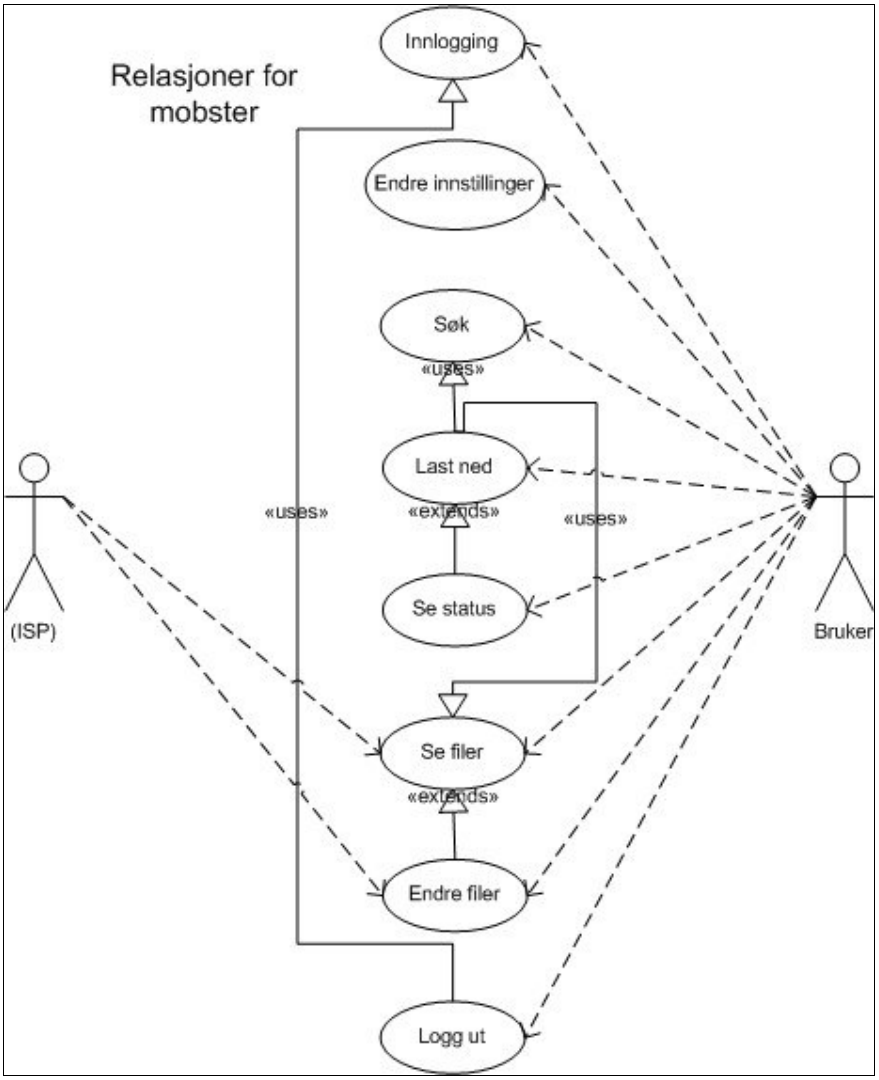
Det er også mulig at en eventuell ISP kan tilby ferdig-konfigurerte løsninger av systemet. Brukeren slipper da å stille inn altfor mye før han tar i bruk systemet.
Mål for systemet:

- Minst mulig skriving
- Minst mulig scrolling
- Innholdet må stå i fokus, unngå effekter og animasjoner
- Lage en brukeropplevelse (logiske sekvenser)

Relasjoner

Figur I.3 viser relasjonene mellom aktører og bruksmønstre UML (Unified Modelling Language) for mobster. Enkelte bruksmønstre er avhengige av hverandre, og dette er synliggjort i figuren. For eksempel er det ikke mulig å se status på en nedlastning, uten at det eksisterer en slik nedlastning.

Aktøren ISP, er satt i parentes, fordi det er usikkert hvilken innflytelse en slik aktør skal ha i systemet.



Figur I.3 relasjoner mellom aktører og bruksmønstre

Forklaring til modellen

Aktørene er skissert som strek-figurer på sidene. Bruksmønstrene er de ovale sirklene. De stiplede linjene angir hvilke bruksmønstre de forskjellige aktørene har tilgang til. Pilene mellom bruksmønstrene angir hvilke bruksmønstre som henger sammen, og hvordan de henger sammen.
<<*extends*>> forteller at det ene bruksmønsteret inngår som en del av et annet. <<*uses*>> (brukes istedenfor ”include”) forteller for eksempel at brukeren ikke kan logge seg ut før han har logget seg inn.

II. Kildekoden

Denne versjonen av kildekoden er den som ble brukt under testing på mobiltelefoner. Koden er kuttet ned til det som er ytterst nødvendig. Alle kommentarer er fjernet. For utdypende informasjon om koden, se Javadoc i vedlegg III.

```
/******************
 * fil: mobster.java
 * versjon: 1.2
 * laget av: Henrik Heiestad
 ******************/

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.util.Hashtable;
import java.util.Vector;
import java.util.Timer;
import java.util.TimerTask;
import javax.microedition.io.*;

import javax.microedition.lcdui.*;

import javax.microedition.rms.RecordStore;
import javax.microedition.rms.RecordStoreException;

import javax.microedition.midlet.MIDlet;

import net.jxta.j2me.PeerNetwork;
import net.jxta.j2me.Message;
import net.jxta.j2me.Element;

public final class mobster extends MIDlet
    implements CommandListener, Runnable {

    private static final String RECORD_STORE_NAME = "jxme-mobster";
    private static final int CONFIG_RECORD_INDEX = 1;
    private static final int FILELIST_RECORD_INDEX = 2;
    private static final int DEFAULT_POLL_INTERVAL = 5;
    private static final int DEFAULT_ALERT_TIMEOUT = 5000;
    private static final int DEFAULT_SCROLL = 3;
    private static final int MAX_TEXTFIELD_SIZE = 256;

    private Display display;
    private Form initForm;
    private Form configForm;
    private Form confirmForm;
    private Form searchFileForm;

    private List fileList;
    private List resultList;
    private List resultSenderFileList;
    private List resultSenderPipeList;
    private Hashtable fileIds = new Hashtable();
    private Vector fileListData = new Vector();

    private int returnQueryId = 0;

    private TextField tfRelayHost;
    private TextField tfRelayPort;
    private TextField tfIdentity;
    private TextField tfPollInterval;
    private TextField searchField;

    private PeerNetwork peer = null;

    private byte[] state = new byte[0];
    private int pollInterval = DEFAULT_POLL_INTERVAL;
    private int pollCounter = 10;
    private Thread pollThread = null;
    private boolean searchSent = false;
    private boolean stopPolling = false;
    private boolean connectInitiated = false;
```

```
private boolean connected = false;
private boolean listening = false;
private String resultName = "";
private int listenQueryId = 0;

private static final String MOBSTERNAME_PREFIX = "JxtaMobsterUserName.";
private static final String MOBSTER_GROUPNAME = "MobGroup";
private static final String MOBSTER_PIPEID = "urn:jxta:uuid-" +
    "59616261646162614E50472050325033" +
    "10696353686172652D5B46696C65436104";

private String UNIK_PIPEID = null;
private int unik_requestId = -1;

public mobster() {
    initForm = new Form("mobster");
    fileList = new List("Mine filer", List.IMPLICIT);
    readFiles();
    readConfig();
    initForm.setTitle("MOBster");

   StringItem inittextfield = new StringItem("Developed by:\n", "HH and Telenor R&D");
    Command cmdOK = new Command("OK", Command.OK, 1);
    Command cmdConnect = new Command("Koble til", Command.SCREEN, 1);
    Command cmdSearch = new Command("Initialiser", Command.SCREEN, 2);
    Command cmdFiles = new Command("Mine filer", Command.SCREEN, 3);
    Command cmdConfig = new Command("Innstillinger", Command.SCREEN, 4);
    Command cmdDefault = new Command("Nullstill", Command.SCREEN, 5);
    Command cmdExit = new Command("Avslutt", Command.EXIT, 5);
   initForm.append(inittextfield);
    initForm.addCommand(cmdConnect);
    initForm.addCommand(cmdSearch);
    initForm.addCommand(cmdFiles);
    initForm.addCommand(cmdConfig);
    initForm.addCommand(cmdDefault);
    initForm.addCommand(cmdExit);
    initForm.setCommandListener(this);
}

public void commandAction(Command c, Displayable d) {
    if (c.getCommandType() == Command.EXIT) {
        destroyApp(true);
        notifyDestroyed();
        return;
    }
    String label = c.getLabel();
    if (d == initForm) {
        if (label.equals("Koble til")) {
            initiateConnect();
        } else if (label.equals("Initialiser")) {

            String url1 = "http://128.39.20.41/Stock.png";
            String url2 = "http://128.39.20.41/bilde.png";
            String url3 = "http://128.39.20.41/App.png";
            String url5 = "http://128.39.20.41/Mobster.png";

            try {
                  loadImages(url1);
                  loadImages(url2);
                  loadImages(url3);
                  loadImages(url5);
            } catch(IOException exx) {
                  showAlert("Error loading images",""+exx,
                      AlertType.ERROR,
                      DEFAULT_ALERT_TIMEOUT,
                      initForm);
                        }
        } else if (label.equals("Mine filer")) {
           viewFiles();
        } else if (label.equals("Innstillinger")) {
            editConfig();
        }  else if (label.equals("Nullstill")) {
            display.setCurrent(confirmForm);
        }
    } else if (d == configForm) {
        if (c.getCommandType() == Command.OK) {
            storeConfig();
            display.setCurrent(initForm);
        } else if (c.getCommandType() == Command.BACK) {
```

```
            configForm = null;
            readConfig();
            display.setCurrent(initForm);
        }
    } else if (d == fileList) {
        if (c.getCommandType() == Command.BACK) {
            display.setCurrent(initForm);
            saveFiles();
        }
        if (label.equals("Info")) {
            infoFile();
        } else if (label.equals("Søk")) {
                        searchLocalFile();
        } else if (label.equals("Endre")) {
            showAlert("Ikke implementert",
                      "Tas med i senere versjoner",
                      AlertType.INFO,
                      DEFAULT_ALERT_TIMEOUT,
                      fileList);
        } else if (label.equals("Slett")) {
            deleteFile();
            saveFiles();
        }
    } else if (d == resultList) {
        if (c.getCommandType() == Command.BACK) {
            display.setCurrent(fileList);
            saveFiles();
        }
        if (label.equals("Last ned")) {
            downloadSelectedFile();

        } else if (label.equals("Info")) {
            resultInfo();
        }
    }else if (d == searchFileForm) {
        if (c.getCommandType() == Command.OK) {
           sendSearch(searchField.getString());
        }
    }
}

public void startApp() {
    display = Display.getDisplay(this);
   if ("".equals(tfIdentity.getString())) {
        editConfig();
    } else {
        display.setCurrent(initForm);
    }

    stopPolling = false;
    pollThread = new Thread(this);
    pollThread.start();
}

public void pauseApp() {
    stopPolling = true;
    pollThread = null;
}

public void destroyApp(boolean unconditional) {
    stopPolling = true;
    pollThread = null;

    storeConfig();
    peer = null;
}

private void editConfig() {
    if (configForm == null) {
        configForm = new Form("Innstillinger");
        configForm.append(tfRelayHost);
        configForm.append(tfRelayPort);
        configForm.append(tfIdentity);
        configForm.append(tfPollInterval);

        Command cmdOK = new Command("OK", Command.OK, 1);
        Command cmdBack = new Command("Tilbake", Command.BACK, 2);
        configForm.addCommand(cmdOK);
        configForm.addCommand(cmdBack);
```

```
            configForm.setCommandListener(this) ;
        }
        display.setCurrent(configForm);
    }

    private void readConfig() {
        String prop = null;

        prop = getAppProperty("RelayHost");
        tfRelayHost = new TextField("Relay host: ",
                                    prop == null ? "128.39.20.27" : prop,
                                    MAX_TEXTFIELD_SIZE,
                                    TextField.ANY);
        prop = getAppProperty("RelayPort");
        tfRelayPort = new TextField("Relay port: ",
                                    prop == null ? "9700" : prop,
                                    MAX_TEXTFIELD_SIZE,
                                    TextField.NUMERIC);
        prop = getAppProperty("Identity");
        tfIdentity = new TextField("Identitet: ",
                                   prop == null ? "3510" : prop,
                                   MAX_TEXTFIELD_SIZE,
                                   TextField.ANY);
        prop = getAppProperty("PollInterval");
        tfPollInterval = new TextField("Poll interval: ",
                                   prop == null ?
                                   Integer.toString(pollInterval) : prop,
                                   MAX_TEXTFIELD_SIZE,
                                   TextField.NUMERIC);
        RecordStore rs = null;
        try {
            rs = RecordStore.openRecordStore(RECORD_STORE_NAME, false);
            byte[] data = rs.getRecord(CONFIG_RECORD_INDEX);
            ByteArrayInputStream bais = new ByteArrayInputStream(data);
            DataInputStream dis = new DataInputStream(bais);
            tfRelayHost.setString(dis.readUTF());
            tfRelayPort.setString(dis.readUTF());
            tfIdentity.setString(dis.readUTF());
            tfPollInterval.setString(dis.readUTF());
            try {
                pollInterval = Integer.parseInt(tfPollInterval.getString());
            } catch (NumberFormatException ex) {
                showAlert("Read Config",
                          "Error parsing poll interval: " +
                          tfPollInterval.getString(),
                          AlertType.WARNING,
                          DEFAULT_ALERT_TIMEOUT,
                          initForm);
            }
            int stateLen = dis.readShort();
            state = new byte[stateLen];
            dis.readFully(state);
        } catch (Exception ex) {

        } finally {
            try {
                if (rs != null) {
                    rs.closeRecordStore();
                }
            } catch (Exception ex) {

            }
        }
    }

    private void storeConfig() {
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        DataOutputStream dos = new DataOutputStream(baos);
        RecordStore rs = null;
        try {
            dos.writeUTF(tfRelayHost.getString());
            dos.writeUTF(tfRelayPort.getString());
            dos.writeUTF(tfIdentity.getString());
            dos.writeUTF(tfPollInterval.getString());

            dos.writeShort(state.length);
            dos.write(state);
            dos.close();
            byte[] data = baos.toByteArray();
```

```
            rs = RecordStore.openRecordStore(RECORD_STORE_NAME, true);
            int recordId = CONFIG_RECORD_INDEX;
            try {
                rs.getRecord(recordId);
                rs.setRecord(recordId, data, 0, baos.size());
            } catch (RecordStoreException rex) {
                recordId = rs.addRecord(data, 0, baos.size());
            }

        } catch (Exception ex) {

        } finally {
            try {
                if (rs != null) {
                    rs.closeRecordStore();
                }
                dos.close();
                baos.close();
            } catch (Exception ex) {

            }
        }
    }

    private void resetConfig() {
        if (connected) {
            disconnect();
        }
        peer = null;
        state = new byte[0];
        try {
            RecordStore.deleteRecordStore(RECORD_STORE_NAME);
        } catch (Exception ex) {
        }

       configForm = null;
       readConfig();
       editConfig();
    }

    private void setupConfirmForm() {
        confirmForm = new Form("Reset innstillinger");
        Command cmdOK = new Command("OK", Command.OK, 1);
        Command cmdBack = new Command("Tilbake", Command.BACK, 2);
            StringItem si = new StringItem(null, "Dette vil nullstille dine innstillinger.
            Vil du fortsette?");
        confirmForm.addCommand(cmdOK);
        confirmForm.addCommand(cmdBack);
        confirmForm.setCommandListener(this);
        confirmForm.append(si);
    }

    private void initiateConnect() {
        if (peer == null) {
            peer = PeerNetwork.createInstance(tfIdentity.getString());
        }
        if (connected || connectInitiated) {
            return;
        }
        connectInitiated = true;
    }

    private boolean connect() {
        connectInitiated = false;
       String host = tfRelayHost.getString();
        int port = 0;

        try {
            port = Integer.parseInt(tfRelayPort.getString());
        } catch (NumberFormatException ex) {
            showAlert("Connect",
                      "Error parsing relay port number: " +
                      tfRelayPort.getString(),
                      AlertType.ERROR,
                      DEFAULT_ALERT_TIMEOUT,
                      initForm);
            return false;
        }
```

```
        String url = "http://" + host + ":" + Integer.toString(port);
        try {
            long startTime, endTime;

            startTime = System.currentTimeMillis();
            state = peer.connect(url, state);
            endTime = System.currentTimeMillis();

            showAlert("Tilkoblet " + url,
                      "Tilkobling tok " +
                      Long.toString(endTime-startTime),
                      AlertType.INFO,
                      DEFAULT_ALERT_TIMEOUT,
                      initForm);
            connected = true;

            String pipeType = PeerNetwork.PROPAGATE_PIPE;
            String returnPipeType = PeerNetwork.UNICAST_PIPE;
           listenQueryId = peer.listen(MOBSTERNAME_PREFIX, MOBSTER_PIPEID, pipeType);

            unik_requestId = peer.create(PeerNetwork.PIPE, tfIdentity.getString(),
               PeerNetwork.UNICAST_PIPE);

        } catch (IOException ex) {
            showAlert("Connect",
                      "Error connecting to relay: " + ex.getMessage(),
                      AlertType.ERROR,
                      Alert.FOREVER,
                      initForm);

            return false;
        }
        display.setCurrent(initForm);
        return true;
    }

    private void disconnect() {
        String pipeType = PeerNetwork.PROPAGATE_PIPE;
        String returnPipeType = PeerNetwork.UNICAST_PIPE;

        try {
            peer.close(MOBSTERNAME_PREFIX, MOBSTER_PIPEID, pipeType);
            connected = false;
            if (UNIK_PIPEID!=null) {
                peer.close(tfIdentity.getString(), UNIK_PIPEID, returnPipeType);
            }
        } catch (IOException ex) {
               showAlert("Disconnect",
                      "Error connecting to relay: " + ex.getMessage(),
                      AlertType.ERROR,
                      Alert.FOREVER,
                      initForm);
        }
    }

    static class StatusUpdate extends TimerTask {
        private StringItem elapsed = null;
        private Gauge status = null;
        private int tick = 0;
        private int max = 0;

        StatusUpdate(StringItem elapsed, Gauge status) {
            this.elapsed = elapsed;
            this.status = status;
            max = status.getMaxValue();
        }

        public void run() {
            elapsed.setText(Integer.toString(++tick) + "s");
            status.setValue(tick % max);
        }
    }

    public void run() {
       while (!stopPolling) {
            if (!connected && connectInitiated) {
                Form connectingForm = new Form("Kobler til...");
                StringItem elapsed = new StringItem("", null);
                connectingForm.append(elapsed);
```

```
            Gauge status = new Gauge(null, false, 10, 0);
            connectingForm.append(status);
            display.setCurrent(connectingForm);
            StatusUpdate updater = new StatusUpdate(elapsed, status);
            new Timer().scheduleAtFixedRate(updater, 1000, 1000);
            try {
                connect();
            } finally {
                updater.cancel();
                display.setCurrent(initForm);
                connectingForm = null;
            }
        }
        try {
            poll();
        } catch (Throwable t) {
            showAlert("Poll",
                      "Error processing message: " + t.getMessage(),
                      AlertType.ERROR,
                      DEFAULT_ALERT_TIMEOUT,
                      initForm);
        }
        try {
            Thread.currentThread().sleep(pollInterval * 1000);
        } catch (InterruptedException ignore) {
        }
    }
}//run

public void poll() {
    if (peer == null || !connected) return;

     //metode for å begrense antall poll
    if (searchSent) pollCounter--;
    if (pollCounter == 0) {
        pollInterval = DEFAULT_POLL_INTERVAL;
        searchSent = false;
        pollCounter = 10;
    }
    Message msg = null;
    try {
        if (peer != null) {
            msg = peer.poll(1);
        }
    } catch (IOException ex) {
        showAlert("Poll",
                  "Error polling relay: " + ex.getMessage(),
                  AlertType.ERROR,
                  DEFAULT_ALERT_TIMEOUT,
                  initForm);
        return;
    }
    if (msg == null) {
        return;
    }
    Element el = null;
    String name = null;
           String rids = null;
           int rid = 0;

    for (int i=0; i < msg.getElementCount(); i++) {
        el = msg.getElement(i);
        if (Message.PROXY_NAME_SPACE.equals(el.getNameSpace())) {
                String elementName = el.getName();

            if (Message.NAME_TAG.equals(elementName)) {
                   name = new String(el.getData());
            } else if (Message.REQUESTID_TAG.equals(elementName)) {
                rids = new String(el.getData());
               try {
                   rid = Integer.parseInt(rids);
               } catch (NumberFormatException nfx) {
                    System.err.println("Recvd invalid " +
                                       Message.REQUESTID_TAG +
                                       ": " + rids);
                    continue;
               }
            } else if (Message.ID_TAG.equals(elementName)) {
                   if(rid == unik_requestId){
```

```
                    UNIK_PIPEID = new String(el.getData());
                   }
                }
              }
            }

    if (name != null) {
        if (name.indexOf(MOBSTERNAME_PREFIX) >= 0) {
            name = name.substring(MOBSTERNAME_PREFIX.length());
        }
    }

    String senderName = null;
    String senderPipe = null;
    String message = null;

    String imageCaption = null;
    String imageFileName = null;
    byte[] imageData = null;

    boolean isDisplayable = true;
    for (int i=0; i < msg.getElementCount(); i++) {
        el = msg.getElement(i);
        if ("JxtaMobsterSenderName".equals(el.getName())) {
            senderName = new String(el.getData());

        } else if ("JxtaMobsterSenderCommand".equals(el.getName())) {
            message = new String(el.getData());
         }  else if ("FileName".equals(el.getName())) {
            imageFileName = new String(el.getData());
         } else if ("Datablock".equals(el.getName())) {
            imageData = el.getData();
            isDisplayable = false;
            lagreSortert(imageFileName, imageData);
            showAlert("Nedlastning ferdig", imageFileName + " er lagret",
           AlertType.INFO,
                     DEFAULT_ALERT_TIMEOUT, fileList);
        } else if ("Sender".equals(el.getName())) {
            senderPipe = new String(el.getData());
        }
        isDisplayable = true;
    }
   if ((senderPipe != null) && (!senderPipe.equals(UNIK_PIPEID))) {
           boolean parseOK = parseCommand(message, senderName, senderPipe);
           senderPipe = null;
    }
}

private void showAlert(String title, String message, AlertType type,
                       int timeout, Displayable back) {
    Alert alert = new Alert(title, message, null, type);
    alert.setTimeout(timeout);
    display.setCurrent(alert, back);
}

private boolean parseCommand(String message, String senderName, String senderPipe){
    String searchString = null;
    String resultString = null;
    String getString = null;
    String errorString = null;
    String response = "RESULT:";
    if (message.indexOf("SEARCHSTRING:") >= 0) {
           searchString = message.substring("SEARCHSTRING:".length());
           if (searchString != null) {
                   Vector results = search(searchString);
                   if (results.size() == 0) {}
                   else {
                       for(int i=0; i<results.size(); i++) {
                           response += (String)results.elementAt(i);
                       }
                       sendResult(response, senderName, senderPipe);
                   }
           }
           return true;
    }

    if (message.indexOf("RESULT:") >= 0) {
        resultString = message.substring("RESULT:".length());
       String resultat = "" + resultString + " : "  + senderName;
```

```
            if (resultString.equals("No files found")) {}
            else {
                resultList.append(resultat, null);
                resultSenderFileList.append(resultString, null);
                resultSenderPipeList.append(senderPipe, null);
                resultList.setTitle("" + resultList.size() + " treff");
                display.setCurrent(resultList);
                resultName = senderName;
            }
             return true;
        }


        if (message.indexOf("GET:") >= 0) {
            getString = message.substring("GET:".length());
             response = "";

            if (getString != null) {

               Vector results = search(getString);
               if (results.size() == 0) {
                       response = "ERROR: Could not find " + getString + "";
                       sendResult(response, senderName, senderPipe);
               } else {
                       for(int i=0; i<results.size(); i++) {
                              sendData((String)results.elementAt(i), senderName,
senderPipe);
                       }
               }
           }
           return true;
        }
        if (message.indexOf("ERROR:") >= 0) {
            errorString = message.substring("ERROR:".length());
                       showAlert("Error",
                      "Fant ikke den etterspurte filen hos " + senderName + "\nPrøv igjen",
                      AlertType.ERROR,
                      DEFAULT_ALERT_TIMEOUT,
                      fileList);
            return true;
        }

        return false;
        }



        private boolean sendResult(String msg, String senderName, String senderPipe) {
        int test = 0;
        if (peer == null || !connected) {
            initiateConnect();
            return false;
        }
        Element[] elm = new Element[4];
        elm[0] = new Element("JxtaMobsterSenderName",
                             tfIdentity.getString().getBytes(),
                             null, null);
        elm[1] = new Element("JxtaMobsterSenderCommand",
                             msg.getBytes(),
                             null, null);
        elm[2] = new Element("Sender",
                             UNIK_PIPEID.getBytes(),
                             null, null);
        elm[3] = new Element("GrpName",
                             "NetPeerGroup".getBytes(),
                             null, null);
        Message m = new Message(elm);

        try {
            String returnPipeType = PeerNetwork.UNICAST_PIPE;
             peer.send(senderName, senderPipe, returnPipeType, m);
        } catch (IOException ex) {
            showAlert("Send",
                      "Error sending message: " + ex.getMessage(),
                      AlertType.ERROR,
                      DEFAULT_ALERT_TIMEOUT,
                      initForm);
            return false;
        }
```

```
        return true;
    }



    void initializeResultList(String title) {
        resultList = new List("Søker etter " + title, List.IMPLICIT);
        resultSenderFileList = new List("", List.IMPLICIT);
        resultSenderPipeList = new List("", List.IMPLICIT);
        Command downloadCommand = new Command("Last ned", Command.SCREEN, 1);
        Command infoCommand = new Command("Info", Command.SCREEN, 2);
        Command backCommand = new Command("Tilbake", Command.BACK, 3);
        resultList.addCommand(downloadCommand);
        resultList.addCommand(infoCommand);
        resultList.addCommand(backCommand);
        resultList.setCommandListener(this);
        display.setCurrent(resultList);
    }

   private boolean sendSearch(String searchString) {
      if (peer == null || !connected) {
            initiateConnect();
            return false;
        }
        initializeResultList(searchString);

        searchSent = true;
        pollInterval = 1;
               String msg = null;
               msg = "SEARCHSTRING:" + searchString;
               if(UNIK_PIPEID==null) {
            showAlert("Error",
                "Programmet har ikke initialisert seg ennå. Prøv igjen senere",
                AlertType.ERROR, DEFAULT_ALERT_TIMEOUT, searchFileForm);
            return false;
        }
       try {
               String MobsterIdentitet = MOBSTERNAME_PREFIX + tfIdentity.getString();
               String returnPipeType = PeerNetwork.UNICAST_PIPE;
               if(!listening) {
                      returnQueryId = peer.listen(MobsterIdentitet, UNIK_PIPEID,
                      returnPipeType);
                      listening = true;
               }
       } catch(IOException ioex) {}

       Element[] elm = new Element[4];
        elm[0] = new Element("JxtaMobsterSenderName",
                             tfIdentity.getString().getBytes(),
                             null, null);
        elm[1] = new Element("JxtaMobsterSenderCommand",
                             msg.getBytes(),
                             null, null);
        elm[2] = new Element("Sender",
                             UNIK_PIPEID.getBytes(),
                             null, null);
        elm[3] = new Element("GrpName",
                             "NetPeerGroup".getBytes(),
                             null, null);
        Message m = new Message(elm);
        try {
            String pipeType = PeerNetwork.PROPAGATE_PIPE;
           String mobsterIdentitet = MOBSTERNAME_PREFIX;
           peer.send(mobsterIdentitet, MOBSTER_PIPEID, pipeType, m);
         } catch (IOException ex) {
            showAlert("Send",
                      "Error sending message: " + ex.getMessage(),
                      AlertType.ERROR,
                      DEFAULT_ALERT_TIMEOUT,
                      initForm);
            return false;
        }
        return true;
    }

    private boolean sendData(String filename, String sender, String senderPipe) {
        if (peer == null || !connected) {
            initiateConnect();
```

```
            return false;
        }
        byte[] getData = null;
        getData = getData(filename);
        if(getData == null) {
            String response = "ERROR: Could not find " + filename + "";
            sendResult(response, sender, senderPipe);
            return false;
        }
        try{

        if (true) {
               Element[] elm = new Element[4];
               elm[0] = new Element("JxtaMobsterSenderName",
                            tfIdentity.getString().getBytes(),
                            null, null);
               elm[1] = new Element("FileName", filename.getBytes(),null,null);

               elm[2] = new Element("Datablock",
                            getData,
                            null, null);

               elm[3] = new Element("GrpName",
                            "NetPeerGroup".getBytes(),
                            null, null);
               Message m = new Message(elm);

               try {
                      String identitet = MOBSTERNAME_PREFIX + sender;
                      String pipeType = PeerNetwork.UNICAST_PIPE;
                      peer.send(identitet, senderPipe, pipeType, m);
               } catch (IOException ex) {
                      showAlert("Send",
                      "Error sending message: " + ex.getMessage(),
                      AlertType.ERROR,
                      DEFAULT_ALERT_TIMEOUT,
                      initForm);
                      return false;
               }
               return true;
               }
        }//try
        catch (Exception e) {

        }
        return true;
    }

       private void loadImages(String url) throws IOException {

       HttpConnection c = null;
       DataInputStream is = null;
       StringBuffer b = new StringBuffer();

       try {
               c = (HttpConnection)Connector.open(url);
                      int status = c.getResponseCode();
                      if (status != 200) {
                             throw new IOException("HTTP Response Code = " + status);
                      }

                      int len = (int)c.getLength();
                      String type = c.getType();
                      String name = c.getFile();

                      if(!type.equals("image/png")) {
                             throw new IOException("Expecting png, recieved " + type);
                      }
                      else {
                             if(len < 1) {
                                    throw new IOException("Lenght is zero or below " +
len);
                             }
                             else {
                                    is = c.openDataInputStream();
                                    byte[] data = new byte[len];
                                    is.readFully(data);
                                lagreSortert(name, data);
                             }
```

```
                }
            }
            finally {
            if(is!= null) {
                    is.close();
            }
            if(c != null) {
                    c.close();
            }
        }
        showAlert("Bilder mottatt",
                "Initialiseringen var vellykket",
                AlertType.INFO, DEFAULT_ALERT_TIMEOUT, initForm);
        }

    private void deleteFile() {
        int i = fileList.getSelectedIndex();
         String utskrift = fileList.getString(i);
        fileList.delete(i);
        fileListData.removeElementAt(i);
        showAlert("Slettet",
                utskrift + " er slettet",
                AlertType.INFO, DEFAULT_ALERT_TIMEOUT, fileList);
    }

    private void resultInfo() {
        int i = resultList.getSelectedIndex();
        String utskrift = "Info om \"" + resultSenderFileList.getString(i) + "\"\nAvsender:
        "+
               resultList.getString(i) + "\nAdresse: " + resultSenderPipeList.getString(i);
        showAlert("Info",
                utskrift,
                AlertType.INFO, DEFAULT_ALERT_TIMEOUT, resultList);
    }

    private void downloadSelectedFile() {
        int i = resultList.getSelectedIndex();
        String utskrift = resultList.getString(i);
        String sendSearch = "GET:" + resultSenderFileList.getString(i);

        if(sendResult(sendSearch, resultName, resultSenderPipeList.getString(i))) {
            showAlert("Forespørsel sendt",
                "Fil: " + resultSenderFileList.getString(i),
                AlertType.INFO, DEFAULT_ALERT_TIMEOUT, resultList);
        }
        else {
            showAlert("Forespørsel ikke sendt",
                "En feil har oppstått: " + resultSenderFileList.getString(i),
                AlertType.ERROR, DEFAULT_ALERT_TIMEOUT, resultList);
        }
    }


    private void lagreSortert(String name, byte[] data) {
         Image source = Image.createImage(data,0,(data.length-1));
         for(int i=0; i<fileList.size(); i++) {
               if (fileList.getString(i).compareTo(name) > 0) {
                       fileList.insert(i, name, source);
                       fileListData.insertElementAt(data, i);
                       return;
               }
         }
         fileList.append(name, source);
          fileListData.addElement(data);
    }

    private void infoFile() {
        int i = fileList.getSelectedIndex();
        String utskrift = fileList.getString(i);
        byte[] tempData = (byte[])fileListData.elementAt(i);
        int picSize = tempData.length;
        showAlert("Info om " + utskrift,
                "filtype: .png\nstørrelse: " + picSize + " byte",
                AlertType.INFO, DEFAULT_ALERT_TIMEOUT, fileList);
     }

    private byte[] getData (String fileName) {
        for(int i=0; i<fileList.size(); i++) {
            if(fileList.getString(i).equals(fileName)) {
```

```
                return (byte[])fileListData.elementAt(i);
            }
         }
         return null;
    }

    private void viewFiles() {
        display.setCurrent(fileList);
    }

    private Vector search(String searchString){
        Vector hits = new Vector();
        for(int i = 0; i < fileList.size(); i++) {
               if (fileList.getString(i).equals(searchString)) {
                      hits.addElement(fileList.getString(i));
               }
        }
        return hits;
    }

    private void searchLocalFile() {
        int i = fileList.getSelectedIndex();
        searchFileForm = new Form("Nytt søk");
        searchField = new TextField("Filnavn: ",
                                    "",
                                    MAX_TEXTFIELD_SIZE,
                                    TextField.ANY);
        searchFileForm.append(searchField);
        Command cmdOK = new Command("Søk", Command.OK, 1);
        searchFileForm.addCommand(cmdOK);
        searchFileForm.setCommandListener(this) ;
        display.setCurrent(searchFileForm);
    }


    private void readFiles() {
        Command cmdChat = new Command("Info", Command.SCREEN, 2);
        Command cmdSort = new Command("Endre", Command.SCREEN, 3);
        Command cmdEdit = new Command("Søk", Command.SCREEN, 1);
        Command cmdDelete = new Command("Slett", Command.SCREEN, 4);
        Command cmdBack = new Command("Tilbake", Command.BACK, 5);
        fileList.addCommand(cmdChat);
        fileList.addCommand(cmdSort);
        fileList.addCommand(cmdEdit);
        fileList.addCommand(cmdDelete);
        fileList.addCommand(cmdBack);
        fileList.setCommandListener(this);
        fileList.setTitle("Mine filer");
   }
}
```

III. Javadoc

Javadoc fra MOBster. Det kan være små avvik mellom kildekode og Javadoc. Årsaken til dette er at alle kommentarer til koden ble fjernet i forbindelse med uttesting av MOBster på mobiltelefoner (det var viktig å gjøre koden så kort som mulig). Som nevnt tidligere i kapittel 6 er alle metodene i MOBster og en hjelpe-klasse (`StatusUpdate`) samlet i èn klasse av effektivitets hensyn.

Class mobster

```
java.lang.Object
  |
  +--javax.microedition.midlet.MIDlet
        |
        +--mobster
```

All Implemented Interfaces:
javax.microedition.lcdui.CommandListener, java.lang.Runnable

---

public final class **mobster**
extends javax.microedition.midlet.MIDlet
implements javax.microedition.lcdui.CommandListener, java.lang.Runnable

---

Inner Class Summary	
(package private) static class	**mobster.StatusUpdate** En klasse som inkrementerer med en for hver gang mobiltelefonen forsøker å koble seg til relèet

Field Summary	
private static int	**CONFIG_RECORD_INDEX**
private javax.microedition.lcdui.Form	**configForm**
private javax.microedition.lcdui.Form	**confirmForm**
private boolean	**connected**
private boolean	**connectInitiated**
private static boolean	**DEBUG**
private static int	**DEFAULT_ALERT_TIMEOUT**
private static int	**DEFAULT_POLL_INTERVAL**
private static int	**DEFAULT_SCROLL**
private javax.microedition.lcdui.Display	**display**
private javax.microedition.lcdui.List	**fileList**

private static int	**FILELIST_RECORD_INDEX**
private java.util.Vector	**fileListData**
private javax.microedition.lcdui.Form	**initForm**
private boolean	**listening**
private int	**listenQueryId**
private static int	**MAX_TEXTFIELD_SIZE**
private static java.lang.String	**MOBSTER_GROUPNAME**
private static java.lang.String	**MOBSTER_PIPEID**
private static java.lang.String	**MOBSTERNAME_PREFIX**
private net.jxta.j2me.PeerNetwork	**peer**
private int	**pollInterval**
private java.lang.Thread	**pollThread**
private static boolean	**QUANTIFY**
private static java.lang.String	**RECORD_STORE_NAME**
private javax.microedition.lcdui.List	**resultList**
private int	**returnQueryId**
private javax.microedition.lcdui.TextField	**searchField**
private javax.microedition.lcdui.Form	**searchFileForm**
private byte[]	**state**
private boolean	**stopPolling**
private javax.microedition.lcdui.TextField	**tfIdentity**
private javax.microedition.lcdui.TextField	**tfPollInterval**
private	**tfRelayHost**

javax.microedition.lcdui.TextField	
private javax.microedition.lcdui.TextField	**tfRelayPort**
private java.lang.String	**UNIK_PIPEID**
private int	**unik_requestId**

Constructor Summary	
mobster() Konstruktør.	

Method Summary	
void	**commandAction**(javax.microedition.lcdui.Command c, javax.microedition.lcdui.Displayable d) Behandler menyvalg fra brukeren.
private boolean	**connect**() Setter opp en forbindelse med JXTA relè på IP-adresse og port angitt i instillingene
private void	**deleteFile**() Sletter valgt fil i "mine filer"
void	**destroyApp**(boolean unconditional) Avslutter applikasjonen
private void	**disconnect**() Kobler ned forbindelsen til JXTA relè.
private void	**downloadSelectedFile**() Laster ned valgt fil i resultatliste
private void	**editConfig**() Viser innstillingene som er lagret, og gir brukeren mulighet til å endre disse
private byte[]	**getData**(java.lang.String fileName) Returnerer et bilde i form av en byte-array fra fileList
private void	**infoFile**() Viser informasjon om valgt fil i fileList
(package private) void	**initializeResultList**(java.lang.String title) Initialiserer en liste som brukes til å ta i mot resultater på søk.
private void	**initiateConnect**() Opretter en instans av PeerNetwork (en peer med navn lik brukernavnet til brukeren)
private void	**lagreSortert**(java.lang.String name, byte[] data) Lagrer data sortert på navn i fileList
private void	**loadImages**(java.lang.String url) Laster ned forhåndsdefinerte bilder fra webtjener.
private boolean	**parseCommand**(java.lang.String message,

	java.lang.String senderName, java.lang.String senderPipe) Analyserer innkommende meldinger og avgjør hva som skal gjøres med disse
void	**pauseApp**() Setter applikasjonen i pause-modus (stopper pollingen og polle-tråden.
void	**poll**() Poller relèet med et fast intervall
private void	**readConfig**() Leser inn innstillinger fra RMS
private void	**readFiles**() Leser alle filer fra RMS inn i fileList.
private void	**resetConfig**() Gjenoppretter de opprinnelige innstillingene
private void	**resultInfo**() Viser informasjon om treff fra søk
void	**run**() Tegner opp brukergrensesnitt og kaller opp connect() og poll()
private void	**saveFiles**() saveFiles lagrer alle filer listet opp i fileList i RMS.
private java.util.Vector	**search**(java.lang.String searchString) search søker etter searchString i fileList.
private void	**searchLocalFile**() searchLocalFile viser et skjermbilde der brukeren kan oppgi ønsket søkestreng
private boolean	**sendData**(java.lang.String filename, java.lang.String sender, java.lang.String senderPipe) Sender binære filer til en mottaker
private boolean	**sendResult**(java.lang.String msg, java.lang.String senderName, java.lang.String senderPipe) Sender respons på innkommende meldinger tilbake til avsender.
private boolean	**sendSearch**(java.lang.String searchString) Sender en søkestreng til alle som lytter på MOBSTER_PIPE_ID.
private void	**setupConfirmForm**() Lager en form som lar brukeren godkjenne gjenoppretting av innstillinger
private void	**showAlert**(java.lang.String title, java.lang.String message, javax.microedition.lcdui.AlertType type, int timeout, javax.microedition.lcdui.Displayable back) Viser en meldinger til brukeren ved hjelp av Alert
void	**startApp**() Starter opp applikasjonen.
private void	**storeConfig**() Skriver innstillingene til RMS (lagrer innstillingene)
private void	**viewFiles**() Viser liste over filer i fileList

Methods inherited from class javax.microedition.midlet.MIDlet	
`getAppProperty, notifyDestroyed, notifyPaused, resumeRequest`	
Methods inherited from class java.lang.Object	
`, clone, equals, finalize, getClass, hashCode, notify, notifyAll, registerNatives, toString, wait, wait, wait`	

Field Detail

RECORD_STORE_NAME
`private static final java.lang.String` **`RECORD_STORE_NAME`**

CONFIG_RECORD_INDEX
`private static final int` **`CONFIG_RECORD_INDEX`**

FILELIST_RECORD_INDEX
`private static final int` **`FILELIST_RECORD_INDEX`**

DEFAULT_POLL_INTERVAL
`private static final int` **`DEFAULT_POLL_INTERVAL`**

DEFAULT_ALERT_TIMEOUT
`private static final int` **`DEFAULT_ALERT_TIMEOUT`**

DEFAULT_SCROLL
`private static final int` **`DEFAULT_SCROLL`**

MAX_TEXTFIELD_SIZE
`private static final int` **`MAX_TEXTFIELD_SIZE`**

---

display
`private javax.microedition.lcdui.Display` **`display`**

---

initForm
`private javax.microedition.lcdui.Form` **`initForm`**

---

configForm
`private javax.microedition.lcdui.Form` **`configForm`**

---

confirmForm
`private javax.microedition.lcdui.Form` **`confirmForm`**

---

searchFileForm
`private javax.microedition.lcdui.Form` **`searchFileForm`**

---

fileList
`private javax.microedition.lcdui.List` **`fileList`**

Alle filer som er lagret i telefonens RMS leses inn i `fileList` ved oppstart av MOBster

---

fileListData
`private java.util.Vector` **`fileListData`**

---

resultList
`private javax.microedition.lcdui.List` **`resultList`**
Alle resultater på søk samles opp i `resultList` og presenteres for brukeren på skjermen

---

returnQueryId
`private int` **`returnQueryId`**

---

tfRelayHost
`private javax.microedition.lcdui.TextField` **`tfRelayHost`**

---

tfRelayPort
`private javax.microedition.lcdui.TextField` **`tfRelayPort`**

---

tfIdentity
`private javax.microedition.lcdui.TextField` **`tfIdentity`**

---

tfPollInterval
`private javax.microedition.lcdui.TextField` **`tfPollInterval`**

---

searchField
`private javax.microedition.lcdui.TextField` **`searchField`**

---

peer
`private net.jxta.j2me.PeerNetwork` **`peer`**

---

state
`private byte[]` **`state`**
Tilstand ved tilkobling.

---

pollInterval
`private int` **`pollInterval`**

---

pollThread
`private java.lang.Thread` **`pollThread`**

---

stopPolling
`private boolean` **`stopPolling`**

---

connectInitiated
`private boolean` **`connectInitiated`**

---

connected
`private boolean` **`connected`**

---

listening
`private boolean` **`listening`**

---

listenQueryId
`private int` **`listenQueryId`**

DEBUG
`private static final boolean` **`DEBUG`**

QUANTIFY
`private static final boolean` **`QUANTIFY`**

MOBSTERNAME_PREFIX
`private static final java.lang.String` **`MOBSTERNAME_PREFIX`**

MOBSTER_GROUPNAME
`private static final java.lang.String` **`MOBSTER_GROUPNAME`**

MOBSTER_PIPEID
`private static final java.lang.String` **`MOBSTER_PIPEID`**

UNIK_PIPEID
`private java.lang.String` **`UNIK_PIPEID`**
Unik `PIPEID` som hver peer får tildelt fra relèet etter å ha kalt metoden `create()`

unik_requestId
`private int` **`unik_requestId`**

Constructor Detail

mobster
`public` **`mobster`**`()`
Konstruktør. Setter opp menyen og forsiden på programmet.

Method Detail

commandAction
```
public void commandAction(javax.microedition.lcdui.Command c,
                          javax.microedition.lcdui.Displayable d)
```
Behandler menyvalg fra brukeren.
Specified by:
`commandAction` in interface `javax.microedition.lcdui.CommandListener`
Parameters:
`c` - kommandoen/menyvalget som brukeren har valgt
`d` - skjermbildet brukeren befinner seg i

startApp
`public void` **`startApp`**`()`
Starter opp applikasjonen. Oppretter en polle-tråd.
Overrides:
`startApp` in class `javax.microedition.midlet.MIDlet`

pauseApp
`public void` **`pauseApp`**`()`
Setter applikasjonen i pause-modus (stopper pollingen og polle-tråden.
Overrides:
`pauseApp` in class `javax.microedition.midlet.MIDlet`

destroyApp

```
public void destroyApp(boolean unconditional)
```

Avslutter applikasjonen

Overrides:

`destroyApp` in class `javax.microedition.midlet.MIDlet`

Parameters:

`unconditional` -

---

editConfig

```
private void editConfig()
```

Viser innstillingene som er lagret, og gir brukeren mulighet til å endre disse

---

readConfig

```
private void readConfig()
```

Leser inn innstillinger fra RMS

---

storeConfig

```
private void storeConfig()
```

Skriver innstillingene til RMS (lagrer innstillingene)

---

resetConfig

```
private void resetConfig()
```

Gjenoppretter de opprinnelige innstillingene

---

setupConfirmForm

```
private void setupConfirmForm()
```

Lager en form som lar brukeren godkjenne gjenoppretting av innstillinger

---

initiateConnect

```
private void initiateConnect()
```

Opretter en instans av PeerNetwork (en peer med navn lik brukernavnet til brukeren)

---

connect

```
private boolean connect()
```

Setter opp en forbindelse med JXTA relè på IP-adresse og port angitt i instillingene

---

disconnect

```
private void disconnect()
```

Kobler ned forbindelsen til JXTA relè. Lukker alle innkommende piper.

---

run

```
public void run()
```

Tegner opp brukergrensesnitt og kaller opp connect() og poll()

Specified by:

`run` in interface `java.lang.Runnable`

---

poll

```
public void poll()
```

Poller relèet med et fast intervall

---

showAlert

```
private void showAlert(java.lang.String title,
                       java.lang.String message,
```

```
                        javax.microedition.lcdui.AlertType type,
                        int timeout,
                        javax.microedition.lcdui.Displayable back)
```

Viser en meldinger til brukeren ved hjelp av Alert

Parameters:

`title` - overskriften på meldingen

`alertText` - meldingsteksten

`type` - hvilken type melding det er (INFO, ERROR, ALARM, CONFIRMATION, WARNING)

`timeout` - hvor lenge meldingen skal vises på skjermen

`back` - skjermbildet som skal vises etter meldingen

See Also:

`Alert(String title, String alertText, Image alertImage, AlertType alertType)`

---

parseCommand

```
private boolean parseCommand(java.lang.String message,
                             java.lang.String senderName,
                             java.lang.String senderPipe)
```

Analyserer innkommende meldinger og avgjør hva som skal gjøres med disse

Parameters:

`message` - innholdet i innkommende melding

`senderName` - brukernavnet til avsender

`senderPipe` - pipe ID til avsender

Returns:

true om analysen førte til at mobiltelefonen måtte utføre operasjoner, false hvis analysen ikke førte til noe.

---

sendResult

```
private boolean sendResult(java.lang.String msg,
                           java.lang.String senderName,
                           java.lang.String senderPipe)
```

Sender respons på innkommende meldinger tilbake til avsender.

Parameters:

`msg` - meldingen som skal sendes

`senderName` - navn på mottakeren av meldingen

`senderPipe` - pipe ID til mottakerens innkommende pipe

---

initializeResultList

```
void initializeResultList(java.lang.String title)
```

Initialiserer en liste som brukes til å ta i mot resultater på søk.

Parameters:

`title` - tittelen på resultatlisten

---

sendSearch

```
private boolean sendSearch(java.lang.String searchString)
```

Sender en søkestreng til alle som lytter på MOBSTER_PIPE_ID.

Parameters:

`searchString` - søkestrengen brukeren har tastet inn

---

sendData

```
private boolean sendData(java.lang.String filename,
                         java.lang.String sender,
                         java.lang.String senderPipe)
```

Sender binære filer til en mottaker

Parameters:
`filename` - navnet på filen som skal sendes
`sender` - navn på mottakeren av filen
`senderPipe` - pipe ID til mottakerens innkommende pipe

---

loadImages

```
private void loadImages(java.lang.String url)
                 throws java.io.IOException
```

Laster ned forhåndsdefinerte bilder fra webtjener. Brukes til å initialisere programmet.
Parameters:
`url` - URL-en til bildet som skal lastes ned

---

deleteFile

```
private void deleteFile()
```

Sletter valgt fil i "mine filer"

---

resultInfo

```
private void resultInfo()
```

Viser informasjon om treff fra søk

---

downloadSelectedFile

```
private void downloadSelectedFile()
```

Laster ned valgt fil i resultatliste

---

lagreSortert

```
private void lagreSortert(java.lang.String name,
                          byte[] data)
```

Lagrer data sortert på navn i fileList
Parameters:
`name` - navnet på filen som skal lagres
`data` - filen representert som en byte-array (byte[])

---

infoFile

```
private void infoFile()
```

Viser informasjon om valgt fil i fileList

---

getData

```
private byte[] getData(java.lang.String fileName)
```

Returnerer et bilde i form av en byte-array fra fileList
Parameters:
`filename` - filnavnet til filen som skal returneres
Returns:
bildet i form av en byte-array (byte[])

---

viewFiles

```
private void viewFiles()
```

Viser liste over filer i fileList

---

search

```
private java.util.Vector search(java.lang.String searchString)
```

search søker etter searchString i fileList. (Foreløpig kun eksakt match).
Parameters:
`searchString` - søkestrengen
Returns:

Vector en vector som inneholder navnet på alle filer som matcher søkestrengen.

---

searchLocalFile

`private void` **`searchLocalFile`**`()`

searchLocalFile viser et skjermbilde der brukeren kan oppgi ønsket søkestreng

---

saveFiles

`private void` **`saveFiles`**`()`

saveFiles lagrer alle filer listet opp i fileList i RMS.

---

readFiles

`private void` **`readFiles`**`()`

Leser alle filer fra RMS inn i fileList. Viser et skjermbilde med alle filer i fileList.

Class mobster.StatusUpdate

```
java.lang.Object
  |
  +--java.util.TimerTask
        |
        +--mobster.StatusUpdate
```

All Implemented Interfaces:

java.lang.Runnable

Enclosing class:

mobster

---

static class **mobster.StatusUpdate**
extends java.util.TimerTask

En klasse som inkrementerer med en for hver gang mobiltelefonen forsøker å koble seg til relèet

Version:

1.0

Author:

JXTA Community

Field Summary	
private javax.microedition.lcdui.StringItem	**elapsed**
private int	**max**
private javax.microedition.lcdui.Gauge	**status**
private int	**tick**

Fields inherited from class java.util.TimerTask
CANCELLED, EXECUTED, lock, nextExecutionTime, period, SCHEDULED, state, VIRGIN

Constructor Summary	
(package private)	**mobster.StatusUpdate**(javax.microedition.lcdui.StringItem elapsed, javax.microedition.lcdui.Gauge status)

Method Summary	
void	**run**()

Methods inherited from class java.util.TimerTask
cancel, scheduledExecutionTime

Methods inherited from class java.lang.Object
, clone, equals, finalize, getClass, hashCode, notify, notifyAll, registerNatives, toString, wait, wait, wait

Field Detail

elapsed

```
private javax.microedition.lcdui.StringItem elapsed
```

status

```
private javax.microedition.lcdui.Gauge status
```

tick

```
private int tick
```

max

```
private int max
```

Constructor Detail

mobster.StatusUpdate

```
mobster.StatusUpdate(javax.microedition.lcdui.StringItem elapsed,
                     javax.microedition.lcdui.Gauge status)
```

Method Detail

run

```
public void run()
```

Overrides:

`run` in class `java.util.TimerTask`

IV. JXME-API

Her er en fullstendig oversikt over klassene (*PeerNetwork*, *Message* og *Element*) som hører til i JXME-API-et. Oversikten er hentet fra JXTA Community sin hjemmeside [59].

net.jxta.j2me
Class Element

```
java.lang.Object
  |
  +--net.jxta.j2me.Element
```

---

public final class **Element**
extends java.lang.Object

This class represents an Element of a JXTA `Message`. A JXTA Message is composed of several Elements.

This is an immutable class.

Constructor Summary
Element(java.lang.String name, byte[] data, java.lang.String nameSpace, java.lang.String mimeType) Construct an Element from its parts.

Method Summary	
byte[]	**getData**() Return the data in the Element.
java.lang.String	**getMimeType**() Return the MIME type of the data in the Element.
java.lang.String	**getName**() Return the name of the Element.
java.lang.String	**getNameSpace**() Return the namespace used by the Element name.
java.lang.String	**toString**() Return a String representation of the Element.

Methods inherited from class java.lang.Object
Equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

Element

```
public Element(java.lang.String name,
               byte[] data,
               java.lang.String nameSpace,
               java.lang.String mimeType)
```

Construct an Element from its parts.

Parameters:

`name` - the name of the Element

`data` - the data that this Element carries. This data is transported across the network as-is.

`nameSpace` - the name space used by the Element. JXTA messages use the "`jxta`" namespace. JXTA for J2ME messages use a private namespace. If namespace is `null`, the default namespace of "" is used.

`mimeType` - the mimeType of the data. If `null`, the default MIME type of "`application/octet-stream`" is assumed.

Method Detail

getName

```
public java.lang.String getName()
```

Return the name of the Element.

Returns:

the Element name

---

getNameSpace

```
public java.lang.String getNameSpace()
```

Return the namespace used by the Element name.

Returns:

the Element namespace

---

getMimeType

```
public java.lang.String getMimeType()
```

Return the MIME type of the data in the Element.

Returns:

the Element MIME type

---

getData

```
public byte[] getData()
```

Return the data in the Element.

Returns:

the Element data

---

toString

```
public java.lang.String toString()
```

Return a String representation of the Element.

Overrides:

`toString` in class `java.lang.Object`

Returns:

a string representation of the Element

net.jxta.j2me
Class Message

```
java.lang.Object
  |
  +--net.jxta.j2me.Message
```

---

public final class **Message**
extends java.lang.Object

This class represents a JXTA Message. A JXTA Message is composed of several `Element`s. The Elements can be in any order, but certain elements are reserved for use by the JXTA Network. These private Elements use a private namespace.

It also defines convenience methods for accessing commonly-used properties for handling responses to the asynchronous operations defined in `PeerNetwork`.

This is an immutable class.

Field Summary	
Static java.lang.String	**ARG_TAG**
Static java.lang.String	**ATTRIBUTE_TAG**
Static java.lang.String	**DEFAULT_MIME_TYPE**
Static java.lang.String	**DEFAULT_NAME_SPACE**
static Message	**EMPTY** An empty Message to send when we have no outgoing message.
Static java.lang.String	**ERROR_TAG**
Static java.lang.String	**ID_TAG**
Static java.lang.String	**JXTA_NAME_SPACE**
Static java.lang.String	**NAME_TAG**
Static java.lang.String	**PROXY_NAME_SPACE**
Static java.lang.String	**REQUEST_CLOSE**
Static java.lang.String	**REQUEST_CREATE**
Static java.lang.String	**REQUEST_LISTEN**
Static java.lang.String	**REQUEST_SEARCH**
Static java.lang.String	**REQUEST_SEND**
Static java.lang.String	**REQUEST_TAG**
Static java.lang.String	**REQUESTID_TAG**

Static java.lang.String	**RESPONSE_ERROR**
Static java.lang.String	**RESPONSE_INFO**
Static java.lang.String	**RESPONSE_MESSAGE**
Static java.lang.String	**RESPONSE_RESULT**
Static java.lang.String	**RESPONSE_SUCCESS**
Static java.lang.String	**RESPONSE_TAG**
Static java.lang.String	**TYPE_TAG**
Static java.lang.String	**VALUE_TAG**

Constructor Summary	
Message(Element[] elms) Construct a Message from an array of Elements.	

Method Summary	
Element	**getElement**(int index) Return the Element contained in this Message at the specified index.
int	**getElementCount**() Return the number of Elements contained in this Message.
int	**getSize**() Returns the size in bytes of this Message.

Methods inherited from class java.lang.Object
Equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

DEFAULT_NAME_SPACE

public static final java.lang.String **DEFAULT_NAME_SPACE**

JXTA_NAME_SPACE

public static final java.lang.String **JXTA_NAME_SPACE**

PROXY_NAME_SPACE

public static final java.lang.String **PROXY_NAME_SPACE**

DEFAULT_MIME_TYPE
public static final java.lang.String **DEFAULT_MIME_TYPE**

REQUESTID_TAG
public static final java.lang.String **REQUESTID_TAG**

TYPE_TAG
public static final java.lang.String **TYPE_TAG**

NAME_TAG
public static final java.lang.String **NAME_TAG**

ID_TAG
public static final java.lang.String **ID_TAG**

ARG_TAG
public static final java.lang.String **ARG_TAG**

ATTRIBUTE_TAG
public static final java.lang.String **ATTRIBUTE_TAG**

VALUE_TAG
public static final java.lang.String **VALUE_TAG**

REQUEST_TAG
public static final java.lang.String **REQUEST_TAG**

RESPONSE_TAG
public static final java.lang.String **RESPONSE_TAG**

ERROR_TAG
public static final java.lang.String **ERROR_TAG**

REQUEST_CREATE
public static final java.lang.String **REQUEST_CREATE**

REQUEST_SEARCH
public static final java.lang.String **REQUEST_SEARCH**

REQUEST_LISTEN
public static final java.lang.String **REQUEST_LISTEN**

REQUEST_CLOSE
public static final java.lang.String **REQUEST_CLOSE**

REQUEST_SEND
public static final java.lang.String **REQUEST_SEND**

RESPONSE_SUCCESS
public static final java.lang.String **RESPONSE_SUCCESS**

RESPONSE_ERROR
public static final java.lang.String **RESPONSE_ERROR**

RESPONSE_INFO

public static final java.lang.String **RESPONSE_INFO**

RESPONSE_RESULT

public static final java.lang.String **RESPONSE_RESULT**

RESPONSE_MESSAGE

public static final java.lang.String **RESPONSE_MESSAGE**

EMPTY

public static final Message **EMPTY**

An empty Message to send when we have no outgoing message. This helps maintain a persistent connection to an HTTP relay.

Constructor Detail

Message

public **Message**(Element[] elms)

Construct a Message from an array of Elements. The supplied Elements are passed along as-is to the relay. Typically, these Elements would hold application data. Internally, JXTA for J2ME may add its own Elements to the Message for routing and other purposes.

Parameters:

elms - an array of elements

Method Detail

getElementCount

public int **getElementCount**()

Return the number of Elements contained in this Message. Usage:

```
for (int i=0; i < msg.getElementCount(); i++) {
  Element el = msg.getElement(i);
  ...
}
```

Returns:

the number of Elements in this Message.

---

getElement

public Element **getElement**(int index)

Return the Element contained in this Message at the specified index. Usage:

```
for (int i=0; i < msg.getElementCount(); i++) {
  Element el = msg.getElement(i);
  ...
}
```

Parameters:

index - specifies the index of the Element to be returned

Returns:

the Elements in this Message at the specified index.

Throws:

ArrayOutOfBoundsException - if the specified index is out of bounds (negative or greater than getElementCount())

---

getSize

public int **getSize**()

Returns the size in bytes of this Message.

net.jxta.j2me

Class PeerNetwork

```
java.lang.Object
  |
  +--net.jxta.j2me.PeerNetwork
```

---

public final class **PeerNetwork**
extends java.lang.Object

This class is an abstraction for the JXTA Network. It specifies the operations that an application can invoke on the JXTA network.

Field Summary	
static java.lang.String	**DEFAULT_GROUP** The group, when not specified, defaults to the NetPeerGroup.
static java.lang.String	**GROUP** Create or Search for a Group.
static java.lang.String	**PEER** Create or Search for a Peer.
static java.lang.String	**PIPE** Create or Search for a Pipe.
static java.lang.String	**PROPAGATE_PIPE** The JXTA Propagte pipe.
static java.lang.String	**UNICAST_PIPE** The JXTA Unicast pipe.

Method Summary	
int	**close**(java.lang.String name, java.lang.String id, java.lang.String type) Close an input Pipe.
byte[]	**connect**(java.lang.String url, byte[] state) Connect to a relay.
int	**create**(java.lang.String type, java.lang.String name, java.lang.String arg) Create a PEER, GROUP or PIPE
static PeerNetwork	**createInstance**(java.lang.String peername) Factory method, used to create an instance of a PeerNetwork.
int	**listen**(java.lang.String name, java.lang.String id, java.lang.String type) Open a Pipe for input.
Message	**poll**(int timeout) Poll the relay for messages addressed to this Peer.
int	**search**(java.lang.String type, java.lang.String query) Search for Peers, Groups or Pipes.
int	**send**(java.lang.String name, java.lang.String id, java.lang.String type, Message data) Send data to the specified Pipe.

Methods inherited from class java.lang.Object
Equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

DEFAULT_GROUP

public static final java.lang.String **DEFAULT_GROUP**

The group, when not specified, defaults to the NetPeerGroup.

UNICAST_PIPE

public static final java.lang.String **UNICAST_PIPE**

The JXTA Unicast pipe. Use for one-to-one communications with a peer.

PROPAGATE_PIPE

public static final java.lang.String **PROPAGATE_PIPE**

The JXTA Propagte pipe. Messages sent to this pipe are propagated to the entire group.

PEER

public static final java.lang.String **PEER**

Create or Search for a Peer.

GROUP

public static final java.lang.String **GROUP**

Create or Search for a Group.

PIPE

public static final java.lang.String **PIPE**

Create or Search for a Pipe.

Method Detail

connect

```
public byte[] connect(java.lang.String url,
                      byte[] state)
               throws java.io.IOException
```

Connect to a relay. A connection to a relay needs to be established before any of the other operations can be invoked.

Parameters:

url - relay URL

state - a byte array that represents the persistent state of a connection to the PeerNetwork. Initially, this would be null. A successful connect(java.lang.String, byte[]) returns this state information which the application is expected to persist and pass it back to connect, if available.

Returns:

see the description for the state parameter

Throws:

java.io.IOException - if the relay is not accesible

---

search

```
public int search(java.lang.String type,
                  java.lang.String query)
           throws java.io.IOException
```

Search for Peers, Groups or Pipes.

Parameters:

type - one of PEER, GROUP or PIPE

query - an expression specifying the items being searched for and also limiting the scope of items to be returned. This is usually a simple regular expression such as, for example, TicTacToe* to search for all entities with names that begin with TicTacToe.

Returns:

query id that can be used to match responses

Throws:

java.io.IOException - if a communication error occurs with the relay or with the JXTA network

create

```
public int create(java.lang.String type,
                  java.lang.String name,
                  java.lang.String arg)
           throws java.io.IOException
```

Create a PEER, GROUP or PIPE

Parameters:

type - one of PEER, GROUP or PIPE

name - the name of the entity being created.

arg - an optional arg depending upon the type of entity being created. For example, for PIPE, this would be the type of PIPE that is to be created. For example, JxtaUniCast and JxtaPropagate are commonly-used values. This parameter can be null.

Returns:

query id that can be used to match responses.

Throws:

java.io.IOException - if a communication error occurs with the relay or with the JXTA network

---

listen

```
public int listen(java.lang.String name,
                  java.lang.String id,
                  java.lang.String type)
           throws java.io.IOException
```

Open a Pipe for input.

Parameters:

name - the name of the Pipe

id - the id of the Pipe. As a convenience feature, the id can be null. If null, the JXTA Proxy will create an id for the Pipe and return it asynchronously in a response message.

An application is expected to use the returned pipe id in future sessions, otherwise the application may end-up listening multiple times to the same pipe, resulting in duplicate messages.

type - the type of the Pipe. JxtaUnicast and JxtaPropagate are two commonly-used values, for example.

Returns:

query id that can be used to match responses

Throws:

java.io.IOException - if a communication error occurs with the relay or with the JXTA network

---

close

```
public int close(java.lang.String name,
                 java.lang.String id,
                 java.lang.String type)
          throws java.io.IOException
```

Close an input Pipe.

Parameters:

name - the name of the Pipe

id - the id of the Pipe. Can be null.

type - the type of the Pipe. JxtaUnicast and JxtaPropagate are two commonly-used values, for example.

Returns:

query id that can be used to match responses

Throws:

java.io.IOException - if a communication error occurs with the relay or with the JXTA network

---

send

```
public int send(java.lang.String name,
                java.lang.String id,
                java.lang.String type,
                Message data)
         throws java.io.IOException
```

Send data to the specified Pipe.

Parameters:
`name` - the name of the Pipe to which the specified `Message` is to be sent.
`id` - the peer or pipe id to which data is to be sent.
`type` - the type of the Pipe. `JxtaUnicast` and `JxtaPropagate` are two commonly-used values, for example.
`data` - a `Message` containing an array of `Element`s which contain application data that is to be sent
Returns:
query id that can be used to match responses, if any
Throws:
`java.io.IOException` - if there is a problem in sending

---

poll

`public Message poll(int timeout) throws java.io.IOException`

Poll the relay for messages addressed to this Peer.

For optimum performance, it is *highly* recommended that this method be called repeatedly until it returns `null`, draining all queued messages before sending out any new messages.

Parameters:
`int` - timeout time in milliseconds to wait for the response. A timeout of `0` means wait forever.
Returns:
a `Message` containing an array of `Element`s containing incoming data
Throws:
`java.io.IOException` - if there is a problem in communicating with the relay

---

createInstance

`public static PeerNetwork createInstance(java.lang.String peername)`

Factory method, used to create an instance of a PeerNetwork.
Parameters:
`peername` - a name that the user would like to give to this Peer. It need not be unique, but it is better for MIDP clients if it is so.
Returns:
an instance of PeerNetwork.